Python for OSINT. 21-day course for beginners

@cyb_detective, May 2023

Who Is This Course For?

I'm primarily doing a course for followers of my Twitter account (https://twitter.com/cyb_detective), in which I post tweets about OSINT (Open Source Intelligence). For those who are professionally involved or just interested in open data investigations and research.

If you use (or plan to use) OSINT tools written in Python, but you're not satisfied with the standard functionality and would like to modify them a bit, this course will help you learn how to do that as quickly as possible.

Also, this course will help you to automate various routine tasks related to investigations: processing data from API, collecting data from websites, collecting search results, working with Internet archives, creating reports and data visualization.

The main goal of the course is not to teach you how to write Python code, but to teach you to spend less time on routine OSINT tasks. So, in addition to code examples, I will also give you links to different services that will help you solve different problems.

This course will also be useful for those who are far from Computer Science and want to raise their technical level a little, try to use Linux, learn to work with the command line and understand different popular IT terms like "JSON", "API", "WHOIS" etc.

Who should avoid this course?

For those who have never done OSINT and are going to do OSINT. This course consists for the most part of specialized topics related to investigation and data collection.

For those who want to learn Python in order to:

- become a really good developer;
- to take the exam to get into university;
- to be interviewed for a job.

This course omits VERY many important things and sometimes even recommends what could have been called bad practice. There are things that don't matter when writing small automations for everyday OSINT tasks, but are extremely important when creating serious team projects.

How to take this course

The first thing I advise you to do is to look at the table of contents, flip through the pages of the book, and clearly decide if this course will be useful to you.

If you've made a clear decision, read one lesson each day thoughtfully and try every day to think about how you could apply what you have learned to your investigations. If you happen to miss a day or even a week, please don't scold yourself for it, but just continue the course day by day.

I also recommend that you try to run all the sample code and try to change something in it.

All the code samples in the book are available in this repository - https://github.com/cipher387/python-for-OSINT-21-days.

This course is distributed completely free of charge. In the beginning I thought about selling it, but since my subscribers are spread all over the planet and have very different income levels, I decided to distribute it without restrictions.

But to strengthen your discipline and motivate you to take it to the end, I recommend you make a small donation.

Free courses people often don't finish until the end, and paying will help you take learning seriously. Also, every donation will motivate me to make new OSINT courses and make them available to people all over the world.

The amount of donation you determine yourself.

For example, if you smoke, then for you the price of the course may be equal to the price of a pack of your favorite cigarettes.

If you drink alcohol, then the cost of a can of beer in the nearest supermarket or a small glass of wine in a restaurant on the next street.
If you like fast food, go with the price of a small burger or package of fries.

You can send a donation via bank card or PayPal:
https://boosty.to/cyb_detective

If for some reason you don't want to send a donation, I would still be very happy if you took this course.

Day 0. Preparing for work

To fully participate in this course, you need internet access and a computer or smartphone with Python and Git installed, or the latest version of a popular browser to run Gitpod (web app providing development environments in your browser) or its analogues (Repl.it, CodePen, CodeAnywhere etc).

I also recommend that you install the Notion app for your computer or phone so that you can mark there each of the 21 days to complete the "complete course task" task. Notion is free for personal use.

If for some reason you don't like the app, you can just read one chapter of the PDF book a day, but I'd still advise you to take a closer look at Notion.
How to install Python?

I won't go into detail on this course, as all readers work on different platforms. I will just give links to instructions for different platforms.

Installation files:

Windows:
https://www.python.org/downloads/windows/
MacOS :
https://www.python.org/downloads/macos/
Linux:
https://www.python.org/downloads/source/

Installation instructions (for different platforms):
https://wiki.python.org/moin/BeginnersGuide/Download

Applications to run Python scripts from your phone:

Android Termux App
https://play.google.com/store/apps/details?id=com.termux&hl=en&pli=1
(use Linux instructions to install)
iOS Pythonista App https://apps.apple.com/us/app/pythonista-3/id1085978097?ls=1

How to install Git?

Git is a version control system. It helps you debug bugs in your code, allowing you to go back to the state "when it all still worked" and to organize work in large teams of programmers (clearly see "who broke everything").

As part of this course, you'll use Git to copy code samples from Github and to install various OSINT tools.

Installation instructions for Windows, Linux and MacOS:

https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

I myself will be using Gitpod in all the examples. It is an online development environment based on Ubuntu (the Linux distribution).

All you need to use it is a browser and a Github/Gitlab/Bitbucket account for authorization.

If reading the instructions on the above links has made you uncomfortable (you can't find the command line, for example), I recommend that you use the Gitpod too (up to 50 hours per month for free).

Is it necessary to take this course for exactly 21 days?

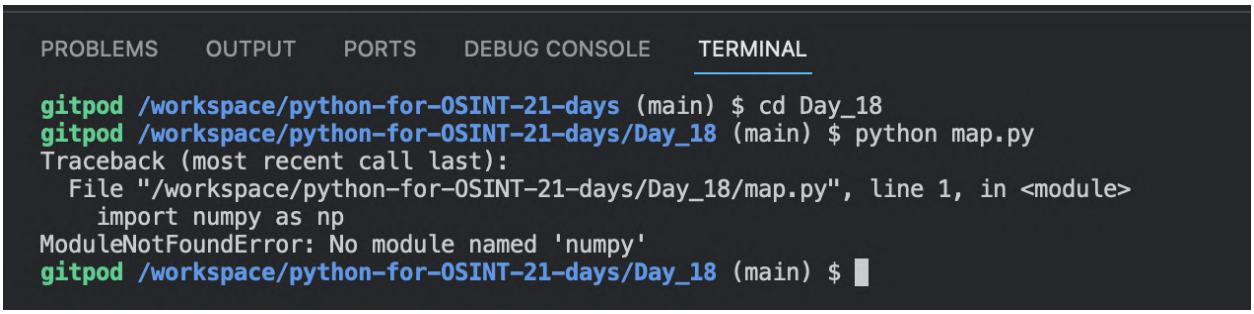
I would strongly advise against doing it any faster. Unless you are a schoolboy (or girl) who is on holiday and has a lot of free time. In that case you can do 2-3 lessons per day (but no more).

If you work 8 or more hours a day, you can do 1 lesson every two or three days. You could also take a break for a few days during the course to give yourself time to rest and reflect on what you've learned.

However, I would not recommend that you do this course for more than two or three months.

I would also recommend that you take the lessons strictly in order.

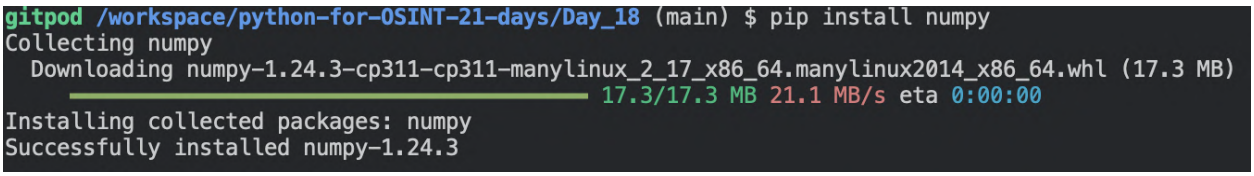
But if you decide otherwise, there is no harm in doing so. Unless you might encounter a "No module named" error if the script uses the package set up in one of the previous lessons.



In this situation, you just need to install the right module (package) using pip.

For example:

pip install numpy



Now let's get to learning!

Day 1. Run the first script

Let's start by copying the Github repository with the sample code files for this course to your computer.

Type in command line:

```
git clone https://github.com/cipher387/python-for-OSINT-21-days
cd python-for-OSINT-21-days
```

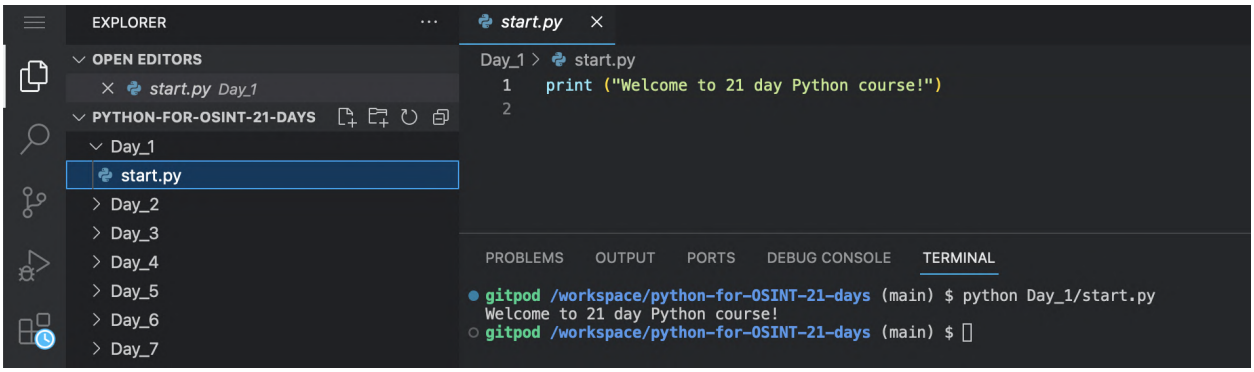
If you see a message asking for your username and password, enter your Github account username and password.

A Github repository is essentially a storage for files. with code, data files, and documentation. It differs from the usual directory in some additional functionality: version history, the ability to make issues (notes with bug reports and questions), forks (copies that are finalized independently of each other) and some other features.

Then, type in the command line:

```
python Day_1/start.py
```

The result should be something like this:



Try changing the text in quotes and run the script again.

If you're not using Gitpod, you may now be faced with the question, "What's the best application to edit Python code files in?"

You can use any text editor you like. Notepad or TextEdit will do too, but I still recommend you to try popular code editors as well: Sublime Text, Notepad++, Visual Studio Code etc. They can automatically highlight syntax and suggest function/variable names (auto-complete code).

That's all for today.

This lesson is much shorter than the others because I want to leave free time for those who haven't had time to install Python yet and for those who already have something going wrong and need to solve some additional problems.

What should you do if something has gone wrong?

Make sure you have installed Python correctly:

```
python –version
```

Make sure you've installed Git accurately:

```
git –version
```

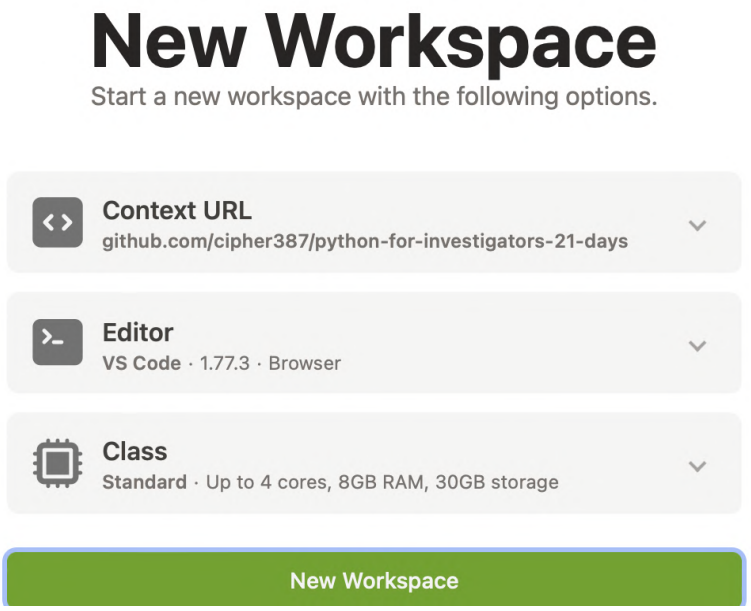
If the folder “python-for-OSINT-21-days” is not copied, check if you entered the password correctly.

If you get an error message when you run the script, try deleting the folder and copying again. Just in case you changed something in the code when you were reviewing it.

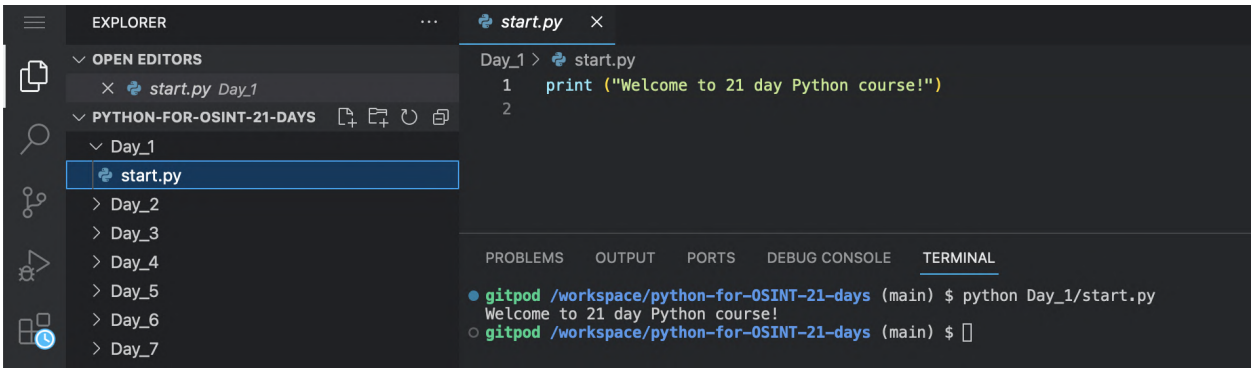
And if none of that helps, I recommend thinking about using Gitpod.

Just open it in your browser:

https://gitpod.io#https://github.com/cipher387/python-for-OSINT-21-days



And create New Workspace with standard settings. If necessary, log into your Github, Gitlab or Bitbucket account.



When it's all ready, type at the command line:

```
python Day_1/start.py
```

Please do not proceed to the next day until you have successfully completed this script and the message "Welcome to 21 day Python course!" appears.

Day 2. Minimum Basic Syntax

Today we're going to introduce you to four basics Python syntax concepts that are also found in most popular programming languages.

I will tell about them as briefly and simple as possible.

Yes, readers who have studied Python before may think I'm missing some very important things. But once again, this course is not intended to make you a good Python developer, it simply shows you possible solutions to the problem of automating a routine in OSINT.

Variable

According to the classical definition, it is a named area of memory that is used to access certain data.

Python variables can store:

text values (e.g., a person's name or a chapter in a book). This type of data is declared with str();
Integer numbers. Declared using the int() function;
Float numbers. Declared using the float() function;
True/false. Declared using the bool() function.

There are a lot of other data types we won't look into in this course.

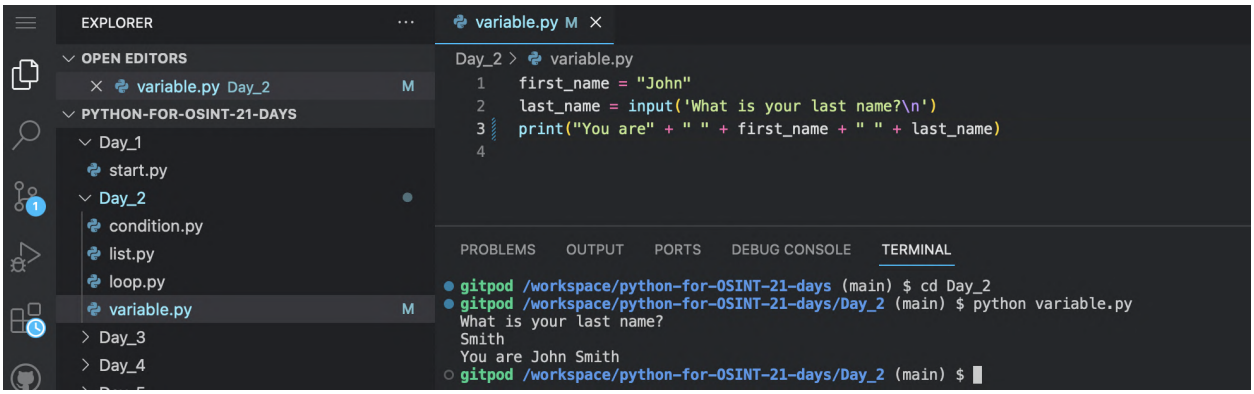
In some languages it is necessary to declare the type of a variable. In Python you don't have to do this unnecessarily (we will only do this a couple of times in this whole course). For example, when you want to append a number to a text string or somehow combine variables that are defined by default as data of different types.

You can use capital letters, small letters, and the underscore character in variable names. You can also use digits, but not as the first character.

Try to give variables names that make as much sense as possible, so that it will be easier for you to understand your code after a while.

Let's practice a little. Run the script variable.py from the Day_2 folder:

```
cd Day_2
python variable.py
```



Note that we run the Python script a little differently than in the first lesson. Last time we specified the path to the file right away, but now we make the Day_2 folder active first, and then run the script. Both variants are acceptable, do it the way you like.

As a result, you should have something similar to the picture.

Hereafter, I will refer to code comments with a # sign. You can also add text after the # sign in script code and it will be ignored by the interpreter.

In this course, I recommend that you first look at the code picture for a couple of minutes and try to guess what the code does. And only after that read the code with comments.

Assign the value John to the first_name variable:

```
first_name = "John"
```

Then we assign to last_name the value entered by the user using input() function. (The \n after the question mark is a line break, you can remove it if you want):

```
last_name = input('What is your last name?\n')
```

And then we output the value of both variables with the function print():

```
print("You are" + " " + first_name + " " + last_name)
```

Note that we use both single and double quotes for text strings. Both types are valid in Python code.

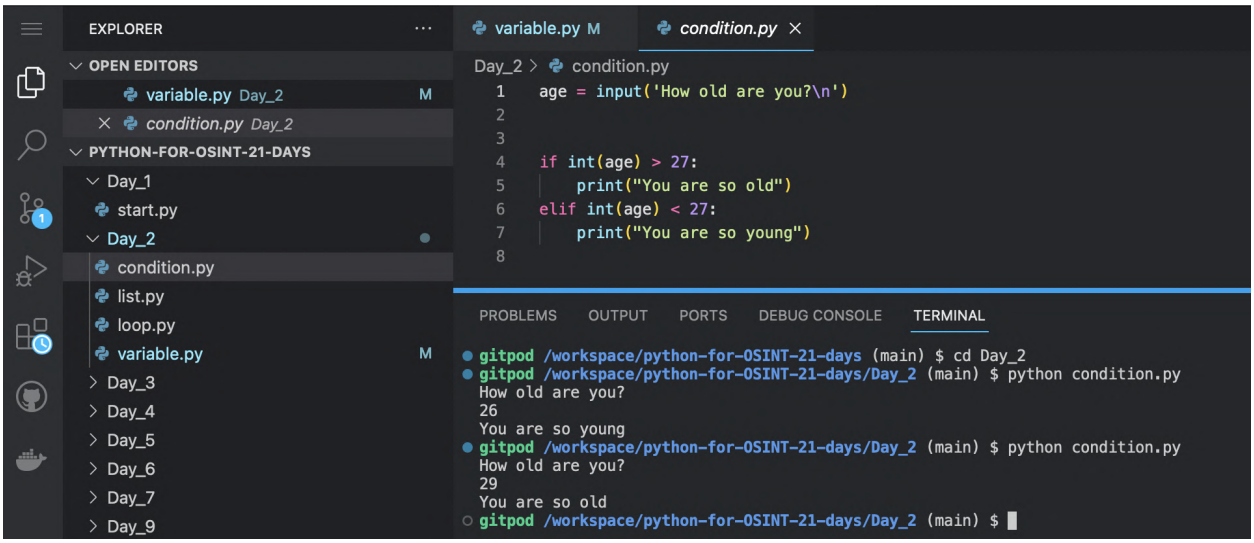
Here we start to use functions. A function is an object that takes arguments as input and returns a certain value or performs a certain action in response.Input() and print() are Python built-in functions that take text strings as arguments.

In a bit of this course, you will learn how to create your own functions.

Conditional statement

This is a syntactic construct that allows you to perform certain actions if a certain condition is met. Let's look at an example right away.

Run condition.py:



First, we use the input() function to ask the user how old he is.

```
age = input('How old are you?\n')
```

If he enters a value greater than 27, we answer that he is very old.

```
if int(age) > 27:
  print("You are so old")
```

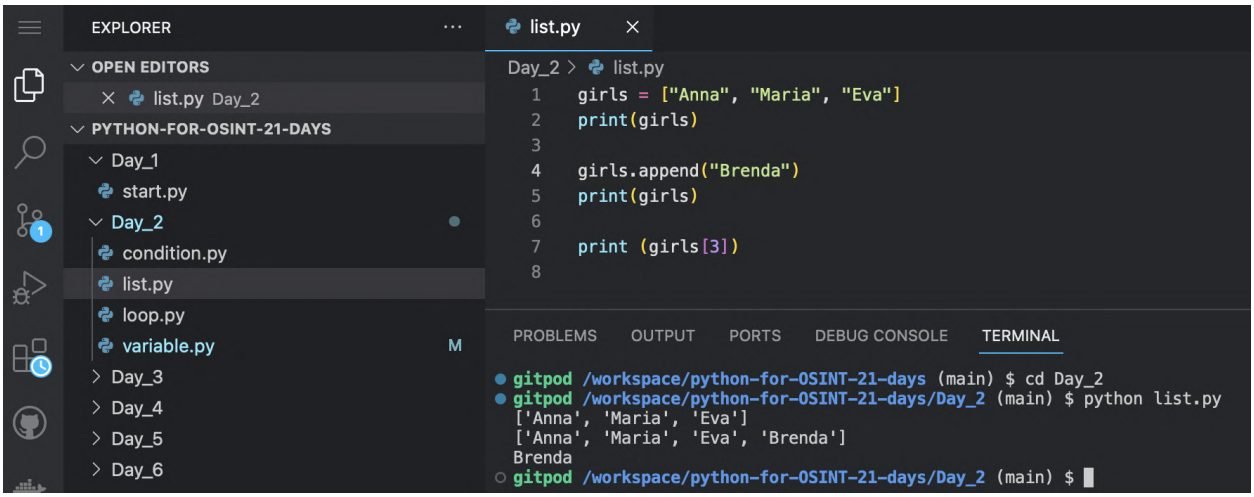
If it is less than 27, he is very young.

```
elif int(age) < 27:
  print("You are so young")
```

List

A list is an ordered set of items, each with its own number or index, allowing quick access to it.

Run list.py:



Create a list of women's names:

```
girls = ["Anna", "Maria", "Eva"]
```

Display it with the print() function:

```
print(girls)
```

Add an item to it with built-in append() function (by default new element will be added to the end of the list):

```
girls.append("Brenda")
```

Print the list:

```
print(girls)
```

Print item number three (the items in the list begin with zero):

```
print (girls[3])
```

In this course we will use lists lots of times and learn lots of different built-in functions for work with them.

If you've studied other programming languages, you're probably familiar with the concept of arrays. Python also has this concept. Python arrays differ from lists in particular by the fact that in lists you can use data of different types (for example, the first element of a list can be a string, and the second a number), while arrays must consist only of numbers or only of strings.

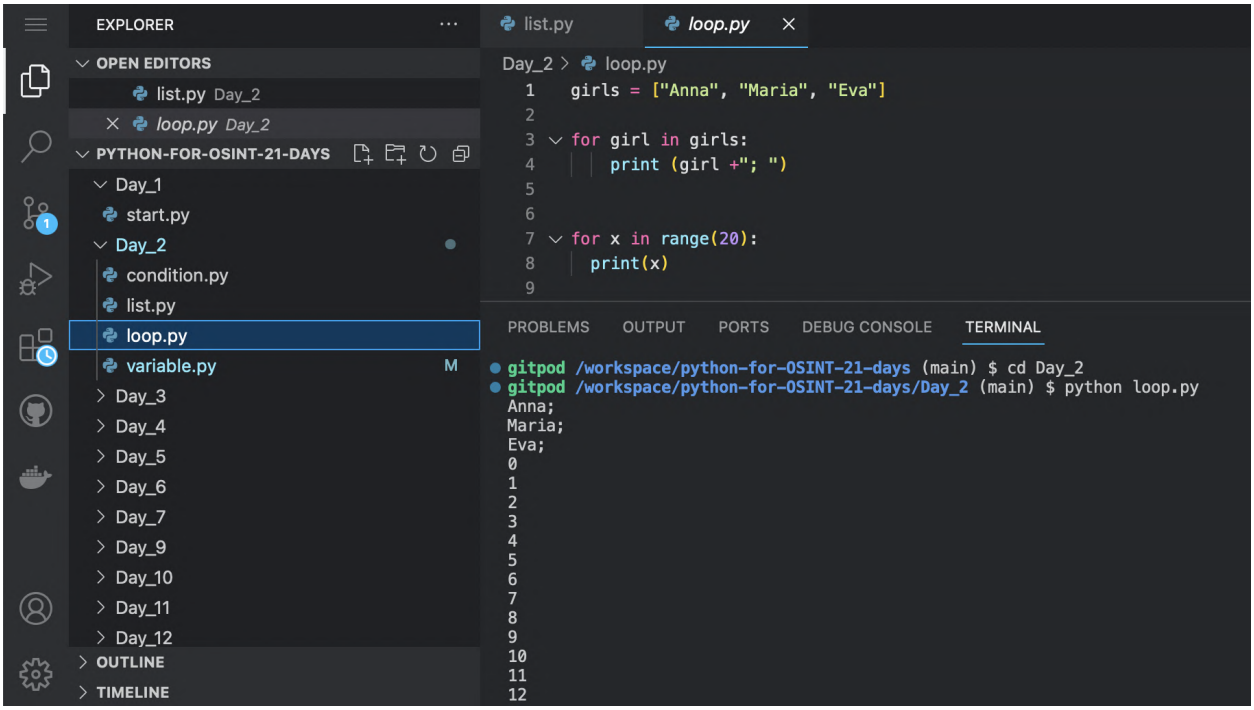
There are other differences that make lists a more flexible and convenient tool. For most tasks related to OSINT, it is sufficient to know how to use lists and we will not study arrays in this course.

Lists can also be multidimensional. When each list item is also a list of 2, 3 or more items. They will be mentioned in the course, but will not receive much attention.

Loop

A loop is a syntactic construct that allows you to repeat a specific code a certain number of times or cycle through all the elements of an array one by one.

Run loop.py:



Create a list of girls' names:

```
girls = ["Anna", "Maria", "Eva"]
```

Print them one by one, adding a semicolon to them:

```
for girl in girls:
    print (girl +"; ")
```

Print numbers from 0 to 19 (remember that counting in Python starts with zero):

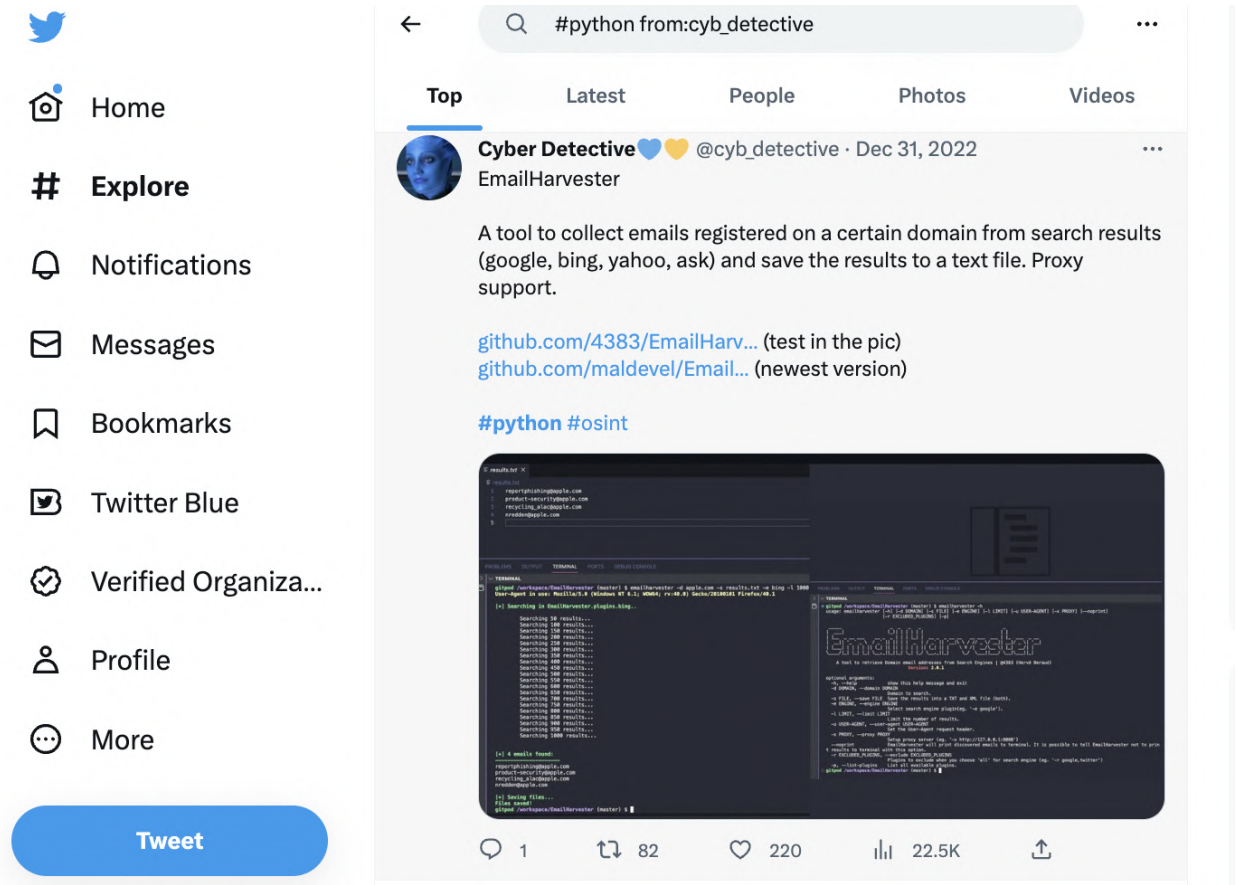
```
for x in range(20):
    print(x)
```

When using loops and conditions, always pay attention to the number of indents. There should always be four spaces before the "internal" code.

In my opinion, this is the minimum theory you need to start writing Python scripts. Tomorrow, we start learning the practical skills that will be useful for OSINT.

Day 3. Install and run Python command line tools

If you often read my Twitter, you may regularly see posts about command line tools for OSINT. Most of them are written in Python. JavaScript (Node.js), Go, Bash (Shell script) and Rust are also popular.



Today we will learn how to set them up to run. As an example, I will use Thorndyke and Blackbird, a tools to search a user's social network pages by nickname.

First way

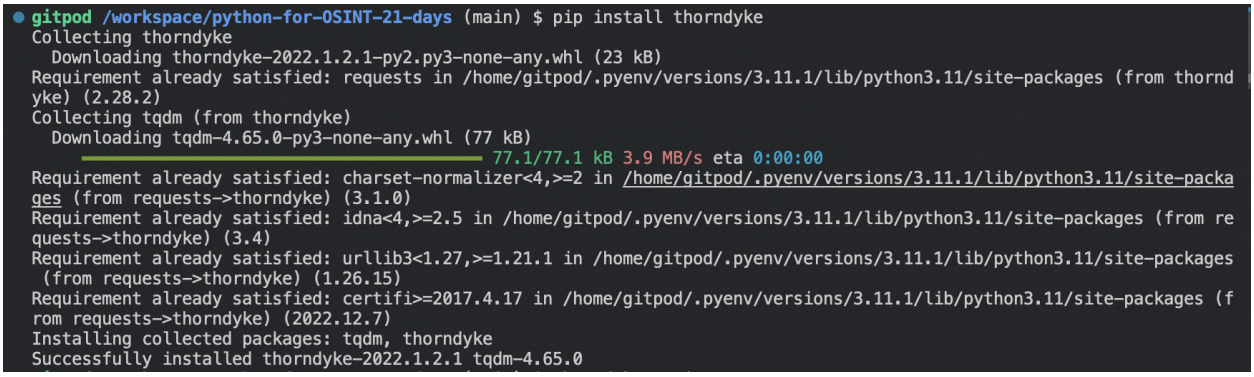
Installation from the Pypi package manager (Package Index).

The Python Package Index (PyPI) is a repository of software for the Python programming language (pypi.org). It contains over 300,000 packages! We will use it in almost every lesson of this course.

Let's start with Thorndyke (https://github.com/rly0nheart/thorndyke), very simple tool to check if a user with a certain nickname exists on various social networks.

Just type at the command line:

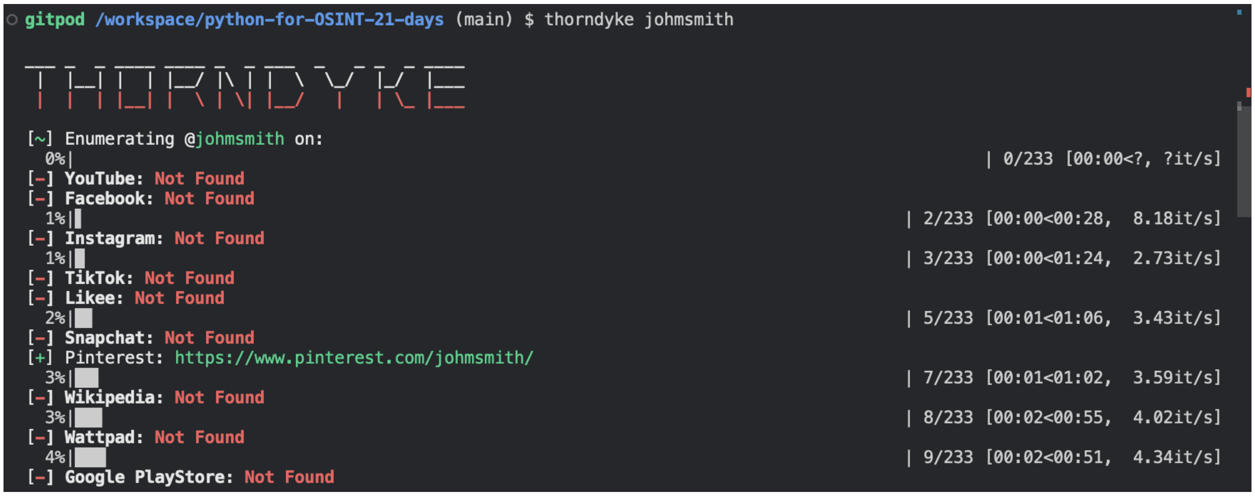
```
pip install thorndyke
```



That's it! The tool can now be run.

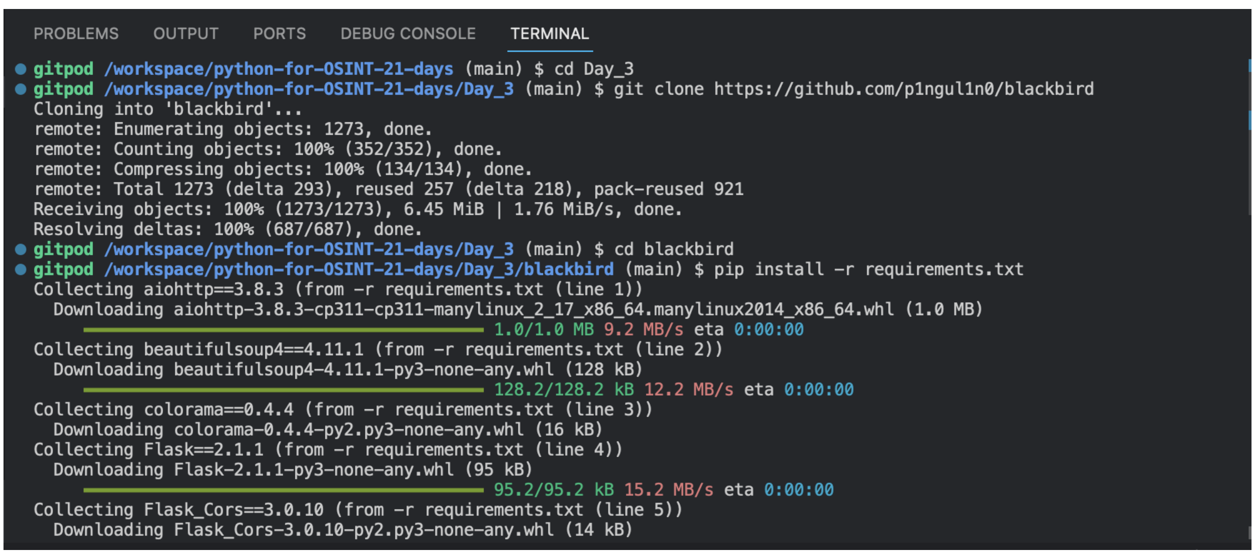
Type thorndyke + the nickname you are interested in at the command line:

```
thorndyke johmsmith
```


gitpod /workspace/python-for-OSINT-21-days (main) $ thorndyke johmsmith
[~] Enumerating @johmsmith on:
0%| | 0/233 [00:00<?, ?it/s]
[-] YouTube: Not Found
[-] Facebook: Not Found
1%| | 2/233 [00:00<00:28, 8.18it/s]
[-] Instagram: Not Found
1%| | 3/233 [00:00<01:24, 2.73it/s]
[-] TikTok: Not Found
[-] Likee: Not Found
2%| | 5/233 [00:01<01:06, 3.43it/s]
[-] Snapchat: Not Found
[+] Pinterest: https://www.pinterest.com/johmsmith/
3%| | 7/233 [00:01<01:02, 3.59it/s]
[-] Wikipedia: Not Found
3%| | 8/233 [00:02<00:55, 4.02it/s]
[-] Wattpad: Not Found
4%| | 9/233 [00:02<00:51, 4.34it/s]
[-] Google PlayStore: Not Found

Second way

Unfortunately, not all tool developers add their projects to PyPi (despite the fact that it's quite easy to do). Therefore, sometimes you have to copy them from Github, install related modules on your own and run the tool by referring directly to the code file instead of the command name.

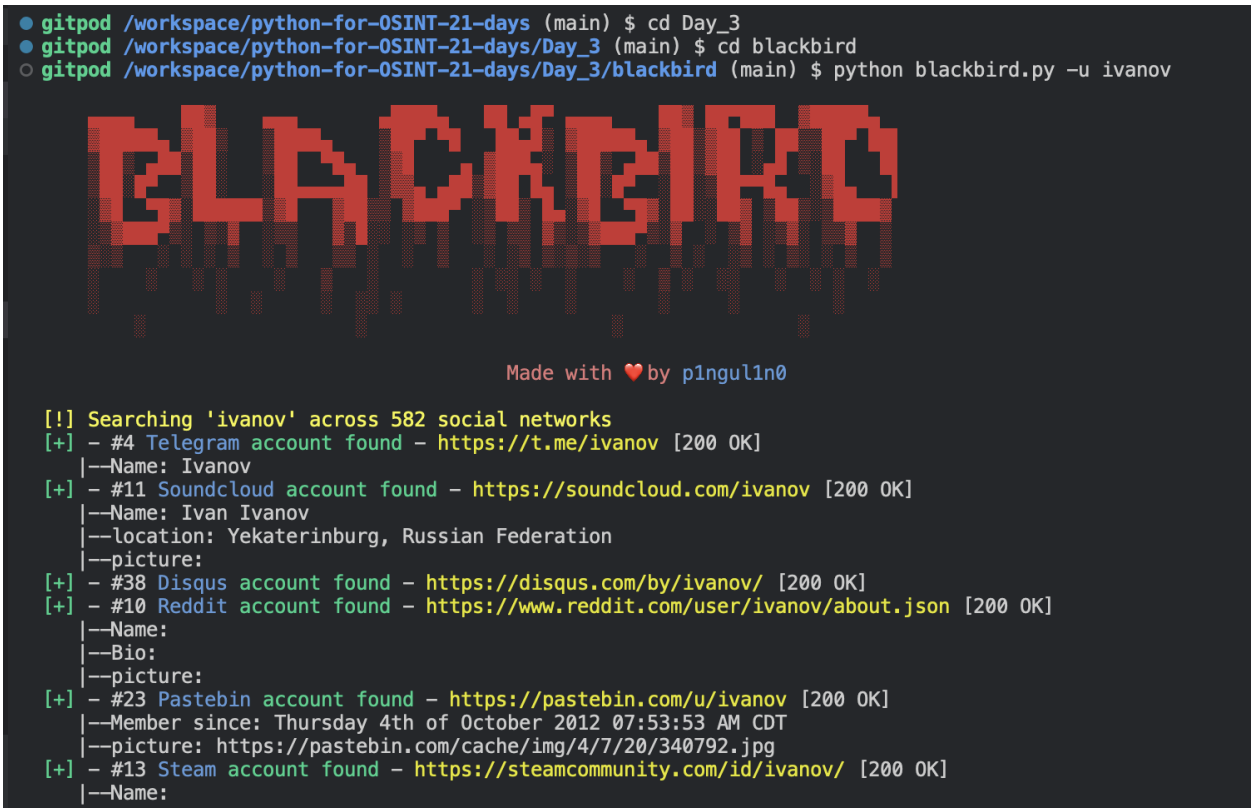

PROBLEMS
OUTPUT
PORTS
DEBUG CONSOLE
TERMINAL
gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_3
gitpod /workspace/python-for-OSINT-21-days/Day_3 (main) $ git clone https://github.com/p1ngul1n0/blackbird
Cloning into 'blackbird'...
remote: Enumerating objects: 1273, done.
remote: Counting objects: 100% (352/352), done.
remote: Compressing objects: 100% (134/134), done.
remote: Total 1273 (delta 293), reused 257 (delta 218), pack-reused 921
Receiving objects: 100% (1273/1273), 6.45 MiB | 1.76 MiB/s, done.
Resolving deltas: 100% (687/687), done.
gitpod /workspace/python-for-OSINT-21-days/Day_3 (main) $ cd blackbird
gitpod /workspace/python-for-OSINT-21-days/Day_3/blackbird (main) $ pip install -r requirements.txt
Collecting aiohttp==3.8.3 (from -r requirements.txt (line 1))
Downloading aiohttp-3.8.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.0 MB)
1.0/1.0 MB 9.2 MB/s eta 0:00:00
Collecting beautifulsoup4==4.11.1 (from -r requirements.txt (line 2))
Downloading beautifulsoup4-4.11.1-py3-none-any.whl (128 kB)
128.2/128.2 kB 12.2 MB/s eta 0:00:00
Collecting colorama==0.4.4 (from -r requirements.txt (line 3))
Downloading colorama-0.4.4-py2.py3-none-any.whl (16 kB)
Collecting Flask==2.1.1 (from -r requirements.txt (line 4))
Downloading Flask-2.1.1-py3-none-any.whl (95 kB)
95.2/95.2 kB 15.2 MB/s eta 0:00:00
Collecting Flask_Cors==3.0.10 (from -r requirements.txt (line 5))
Downloading Flask_Cors-3.0.10-py2.py3-none-any.whl (14 kB)

Now we will install another tool for searching social networking pages by nickname: blackbird (https://github.com/p1ngul1n0/blackbird). Type in the command line:

```
cd Day_3
git clone https://github.com/p1ngul1n0/blackbird
cd blackbird
pip install -r requirements.txt
```

The requirements.txt file contains a list of packages needed to run the tool.

Let me remind you once again, that command "cd" is used to navigate to another folder.



To check if the installation was successful, try running the tool:

```
python blackbird.py -u ivanov
```

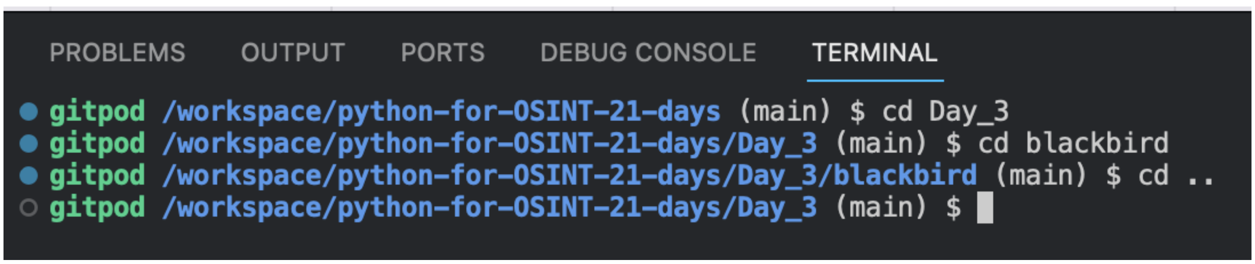
Third way

You can also launch tools directly from Python script code, using Subprocess module (https://docs.python.org/3/library/subprocess.html), which allows you to run different command line commands directly in Python code

Move the launch.py file from the Day_3 folder to the blackbird folder.

```
mv launch.py blackbird
```

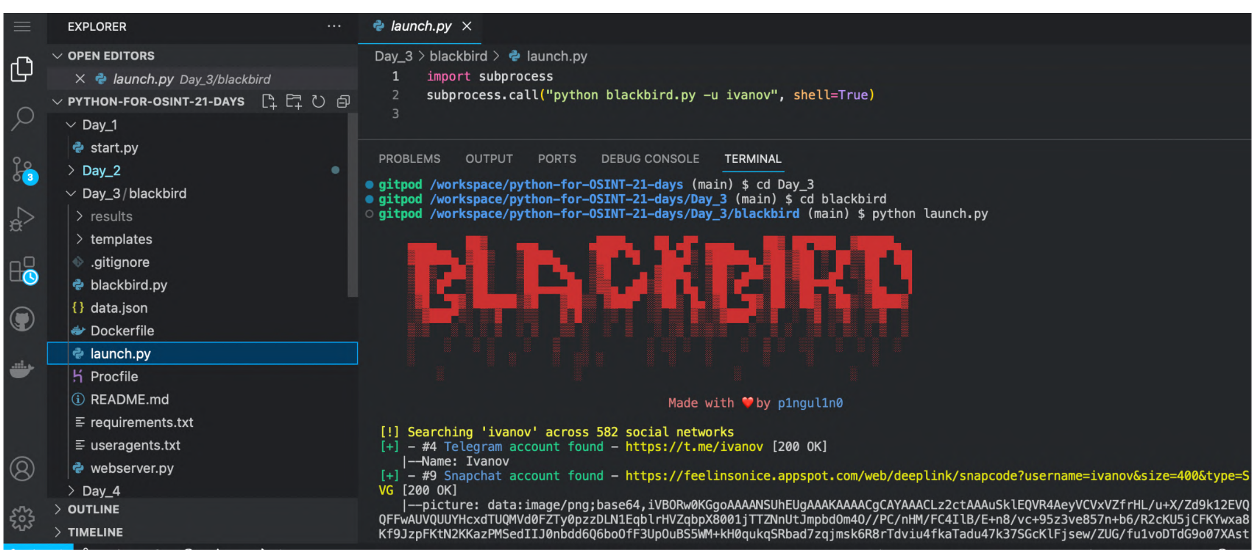
Before running the command, make sure that you are in the Day_3 folder.



If you are in the blackbird folder, use this command to go one directory above:

```
cd ..
```

Run **launch.py**:



```
# Import the subprocess module:

import subprocess

# And run blackbird.py:

subprocess.call("python blackbird.py -u ivanov", shell=True)
```

This way you can run not only Python scripts, but also scripts created in other programming languages.

The most important thing you should remember from this course is that Blackbird and Thorndyke are NOT the best solutions for nickname enumeration.

Sherlock (https://github.com/sherlock-project/sherlock) and Maigret (https://github.com/soxoj/maigret) check much more sites. Try installing and running them yourself.

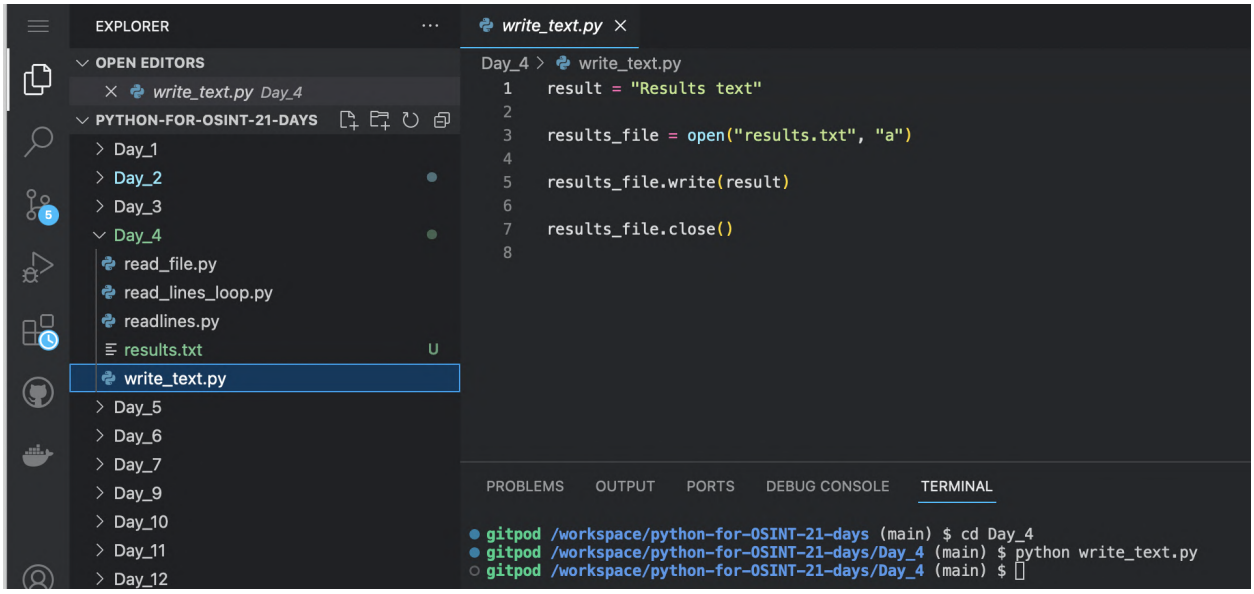
Day 4. Reading and writing files

In the comments to the tweets in @cyb_detective Twitter about command line tools you can often see the comments “Why you need it when there is a web application that has similar functions?”.

One of the advantages of using command line tools in investigations is the ability to save results to files so that they can be automatically analyzed later. In this lesson we will learn how to read and write data from text files using standard Python functions. Let's start with writing.

Writing file

Run **write_text.py**:



Create a variable with a certain text:

```
result = "Results text"
```

Open (and at the same time create) the file results.txt:

```
results_file = open("results.txt", "a")
```

Write the value of the variable into it:

```
results_file.write(result)
```

Close the file:

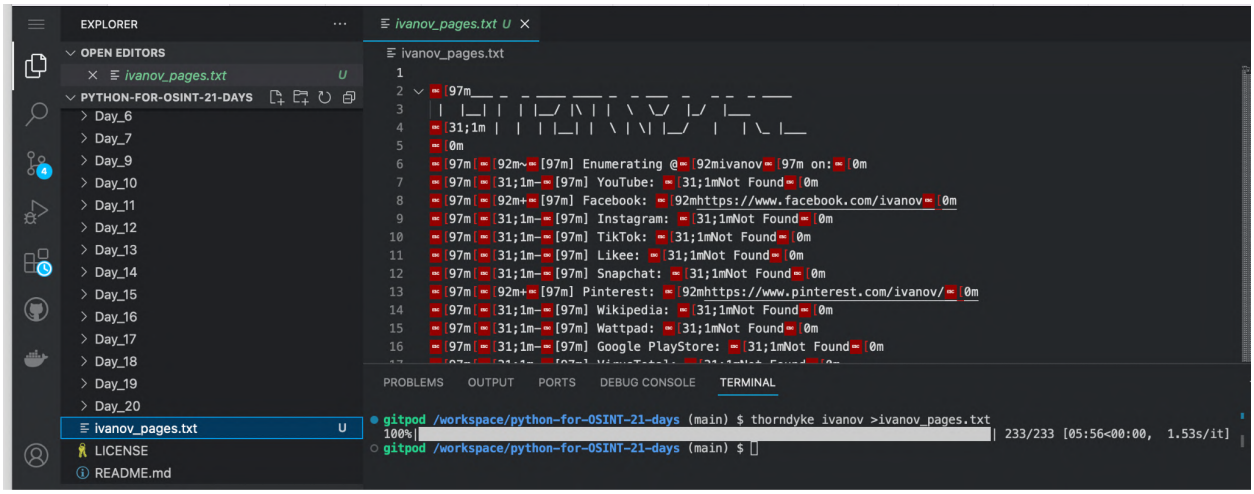
```
results_file.close()
```

Note that the open() function has two arguments. The first is the name of the file and the second is the so-called "opening mode".

Examples of file opening modes:

“r” - open a file for reading.
“a” - open a file for appending (create the file if it does not exist)
“w” - open a file for writing (create the file if it does not exist)
“x” - create new file.

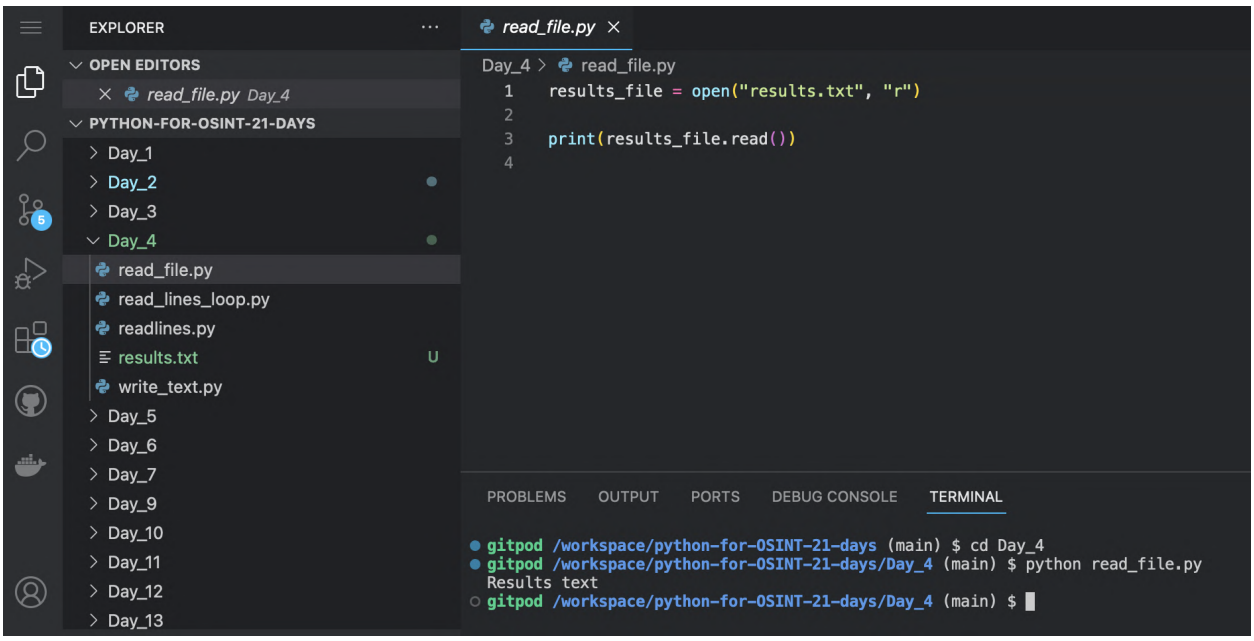
Remember that in some situations it is not necessary to write special code to write the results of the script into a file, because the easiest way to write the results of a Python script to a file is simply to add > and the file name to the command to run it:



Now let's try to read the text of the file we just created.

Reading file

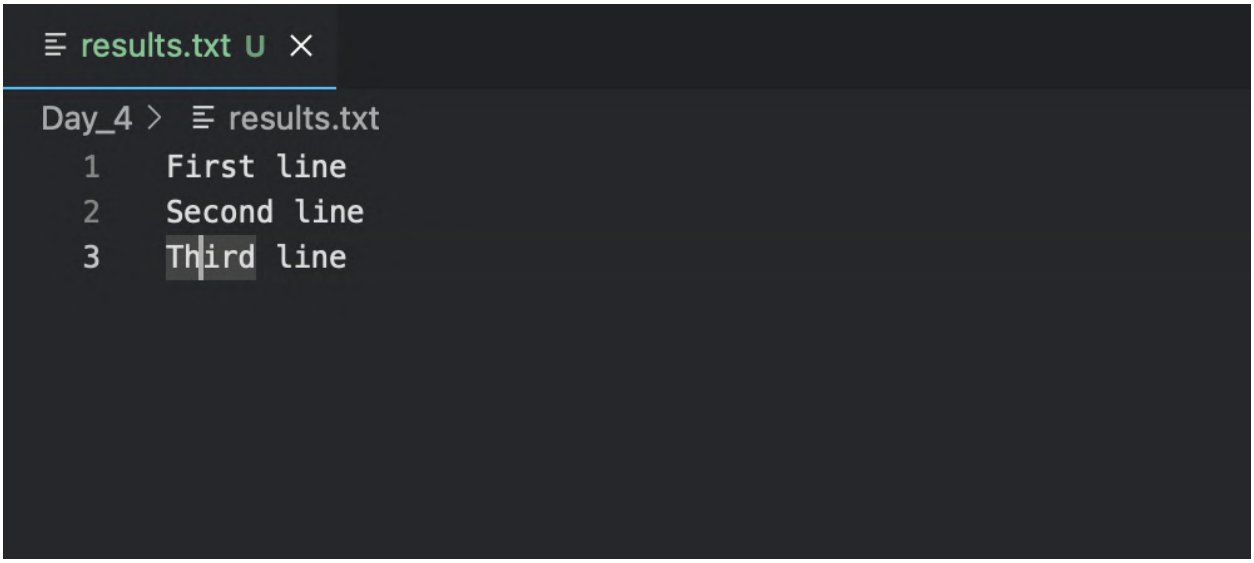
Run **read_file.py**:



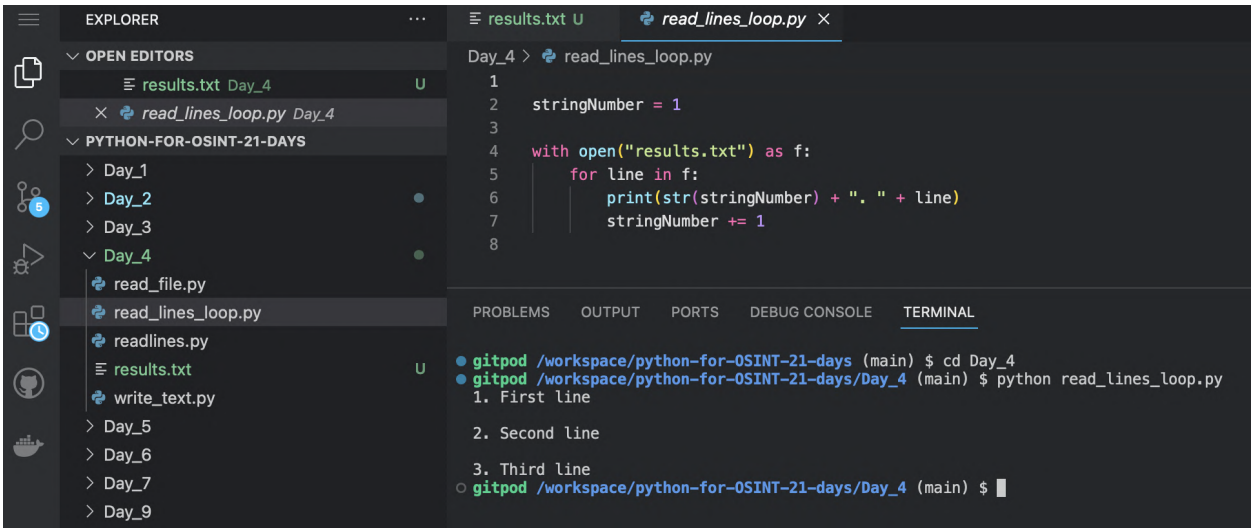
Open the results.txt file:

```
results_file = open("results.txt", "r")

# Display the contents of the results.txt file:

print(results_file.read())
```

There is another way to go. With a simple loop, you can read the lines of a file one at a time and perform some action on each line.



Please, add some strings to results.txt file and run read_lines_loop.py:



Create a variable with the line number:

```
stringNumber = 1
```

Open results.txt file:

```
with open("results.txt") as f:
```

Go through lines and print each line with line number:

```
    for line in f:
        print(str(stringNumber) + ". " + line)
```

Increase the line number by one:

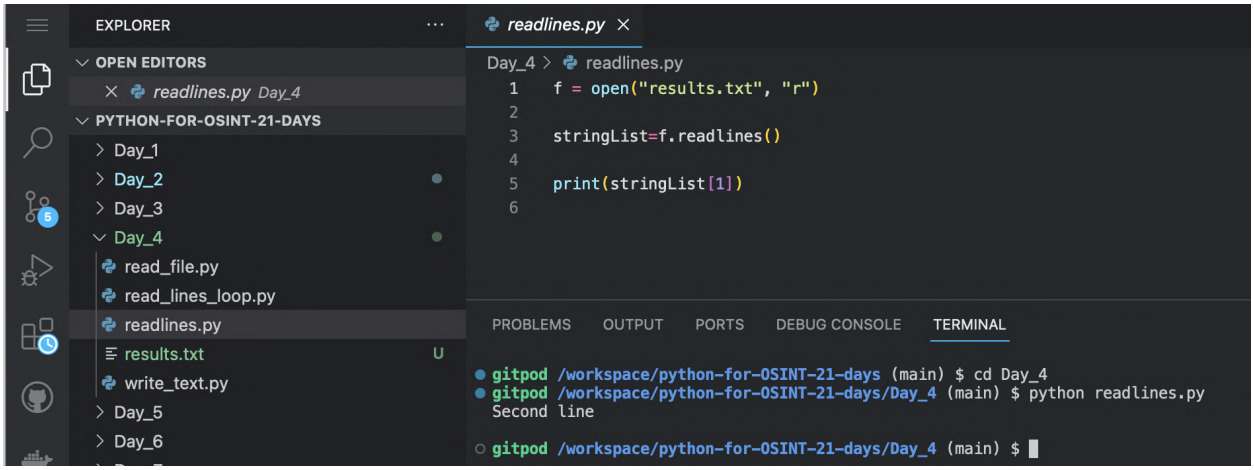
```
        stringNumber += 1
```

Note that we use str() to convert a variable of type integer to a string. You should always do this when you concatenate a text variable and a number into one string.

If you do not want to print all lines in the file, but only lines with certain numbers, you can use the readlines() function, which converts the file lines into list items.

.

Run readlines.py:



.

Open results.txt file:

```
f = open("results.txt", "r")
```

Create an array whose elements are the lines of the results.txt file:

```
stringList=f.readlines()
```

Print the array element with index one (the second line of the file). Don't forget that in lists the counting goes from zero:

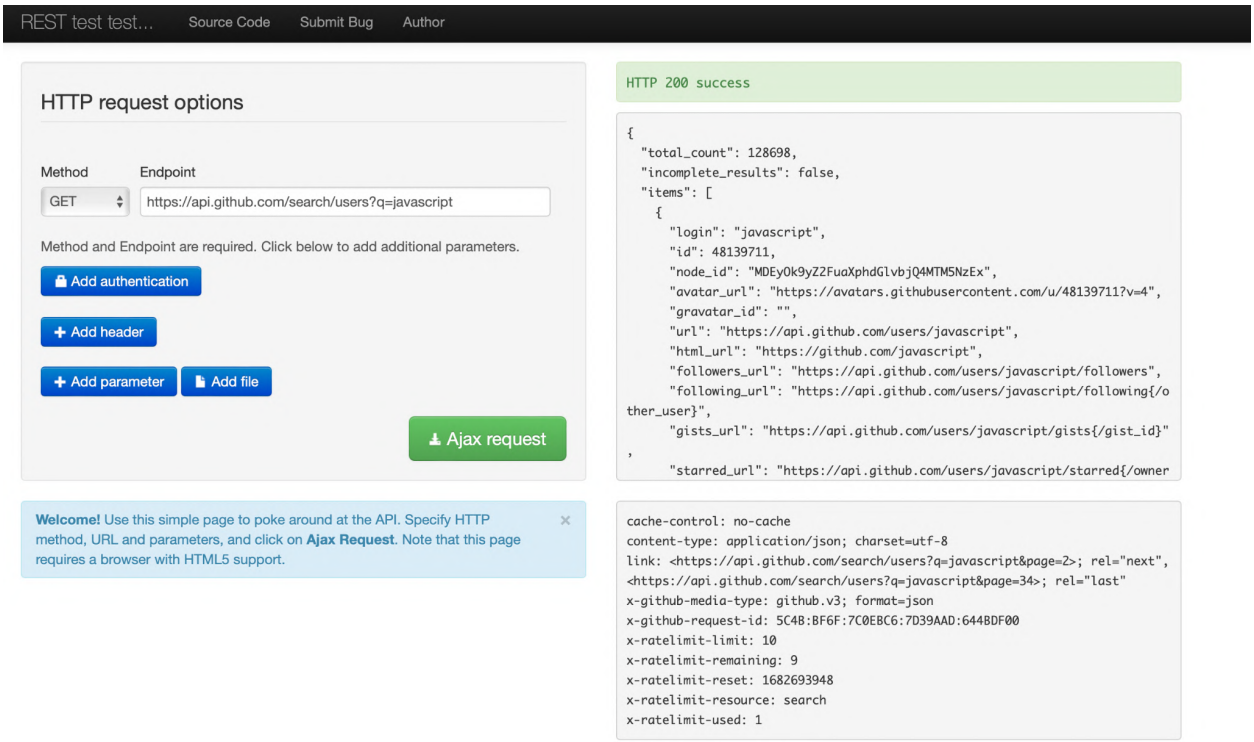
```
print(stringList[1])
```

If, on the contrary, you need to write the array elements to a file, so that each element is written on a separate line, use the writelines() method.

Storing data in files is not always the best practice (although it is the easiest to learn). If you regularly have to work with files that are tens or hundreds of megabytes in size, you should consider starting to use databases. We'll touch on this topic a bit in Lesson 8.

Day 5. Handling HTTP requests and working with APIs

When you open a web page in your browser, there is a request to the server. In response to the request, the server returns the status, headers and body of the response (for example, html-code of web page, some data in CSV, JSON or XML format).



The easiest way to understand what's going on is visually. Open https://resttesttest.com , copy some link there and click the “AJAX request “ button.

OSINT often needs to automate data collection from web pages or various APIs (Application Programming Interface). And the basic skill needed to do this is to write code to send requests to web servers and process the responses.

APIs (Application Programming Interface) is a technology that allows you to interact with an application by sending requests to a server. For example, the Github API allows you to retrieve data about Github users, as well as make changes to repositories and more.

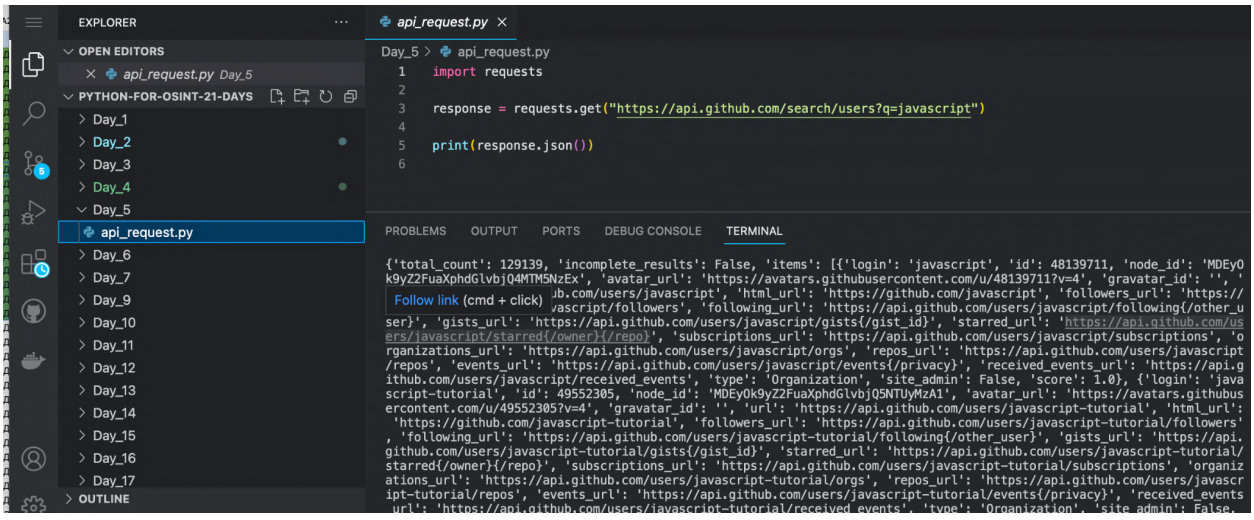
For this we will use packages requests (https://pypi.org/project/requests/).

```
gitpod /workspace/python-for-OSINT-21-days (main) $ pip install requests
Requirement already satisfied: requests in /workspace/.pyenv_mirror/user/current/lib/python3.11/site-packages (2.28.1)
Requirement already satisfied: charset-normalizer<3,>=2 in /workspace/.pyenv_mirror/user/current/lib/python3.11/site-pac
kages (from requests) (2.1.1)
Requirement already satisfied: idna<4,>=2.5 in /home/gitpod/.pyenv/versions/3.11.1/lib/python3.11/site-packages (from re
quests) (3.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /home/gitpod/.pyenv/versions/3.11.1/lib/python3.11/site-packages
 (from requests) (1.26.15)
Requirement already satisfied: certifi>=2017.4.17 in /home/gitpod/.pyenv/versions/3.11.1/lib/python3.11/site-packages (f
rom requests) (2022.12.7)
```

Install request package:

```
pip install requests
```

Run api_request.py:



Add the requests package to the script file using the import command:

```
import requests

# Making a request:

response =
requests.get("https://api.github.com/search/users?q=javascript")

# Display the result in JSON format:

print(response.json())
```

There are a huge number of APIs, both paid and free, which provide useful data for OSINT. For example:

information about telephone numbers, addresses, and zip codes;
information about companies;
Information about domains and IP-addresses;
Information about crypto wallets and transaction;
Information about users of different social media.

A list of over a hundred useful OSINT APIs can be found in this Github repository:

https://github.com/cipher387/API-s-for-OSINT

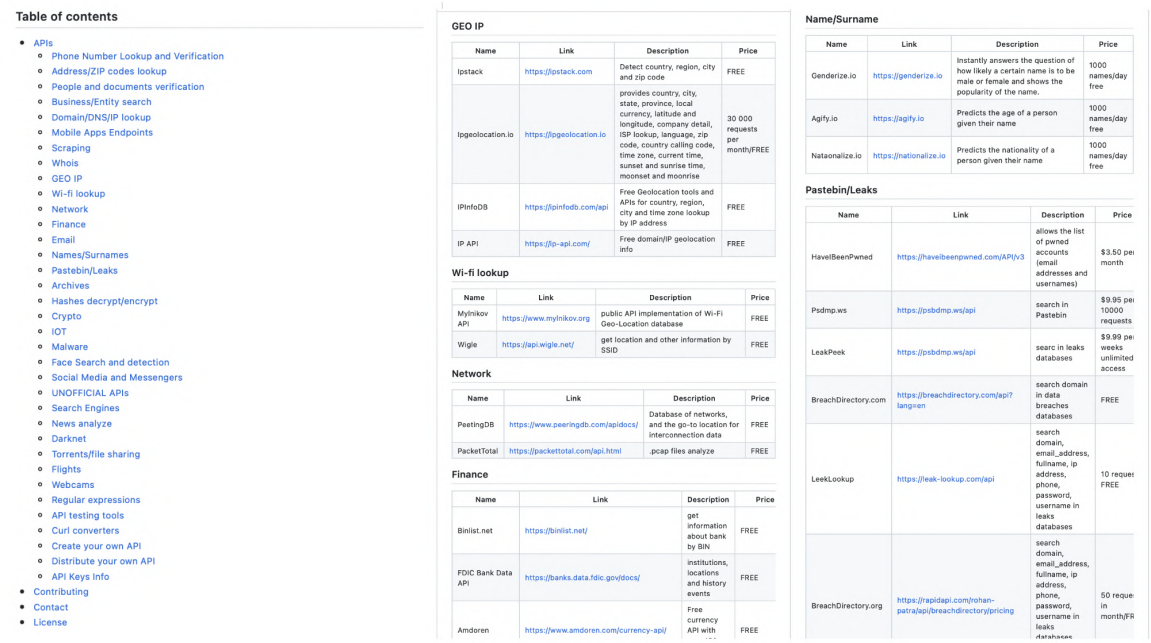


Table of contents

- APIs
  - Phone Number Lookup and Verification
  - Address/ZIP codes lookup
  - People and documents verification
  - Business/Entity search
  - Domain/DNS/IP lookup
  - Mobile Apps Endpoints
  - Scraping
  - Whois
  - GEO IP
  - Wi-fi lookup
  - Network
  - Finance
  - Email
  - Names/Surnames
  - Pastebin/Leaks
  - Archives
  - Hashes decrypt/encrypt
  - Crypto
  - IOT
  - Malware
  - Face Search and detection
  - Social Media and Messengers
  - UNOFFICIAL APIs
  - Search Engines
  - News analyze
  - Darknet
  - Torrents/file sharing
  - Flights
  - Webcams
  - Regular expressions
  - API testing tools
  - Curl converters
  - Create your own API
  - Distribute your own API
  - API Keys Info
- Contributing
- Contact
- License

GEO IP

Name	Link	Description	Price
Ipstack	https://ipstack.com	Detect country, region, city and zip code	FREE
Ipgeolocation.io	https://ipgeolocation.io	provides country, city, state, province, local currency, latitude and longitude, company detail, ISP lookup, language, zip code, country calling code, time zone, current time, sunset and sunrise time, moonset and moonrise	30 000 requests per month/FREE
IPInfoDB	https://ipinfodb.com/api	Free Geolocation tools and APIs for country, region, city and time zone lookup by IP address	FREE
IP API	https://ip-api.com/	Free domain/IP geolocation info	FREE

Wi-fi lookup

Name	Link	Description	Price
Mylnikov API	https://www.mylnikov.org	public API implementation of Wi-Fi Geo-Location database	FREE
Wigle	https://api.wigle.net/	get location and other information by SSID	FREE

Network

Name	Link	Description	Price
PeeringDB	https://www.peeringdb.com/apidocs/	Database of networks, and the go-to location for interconnection data	FREE
PacketTotal	https://packettotal.com/api.html	.pcap files analyze	FREE

Finance

Name	Link	Description	Price
Binlist.net	https://binlist.net/	get information about bank by BIN	FREE
FDIC Bank Data API	https://banks.data.fdic.gov/docs/	institutions, locations and history events	FREE
Amdoren	https://www.amdoren.com/currency-api/	Free currency API with	FREE

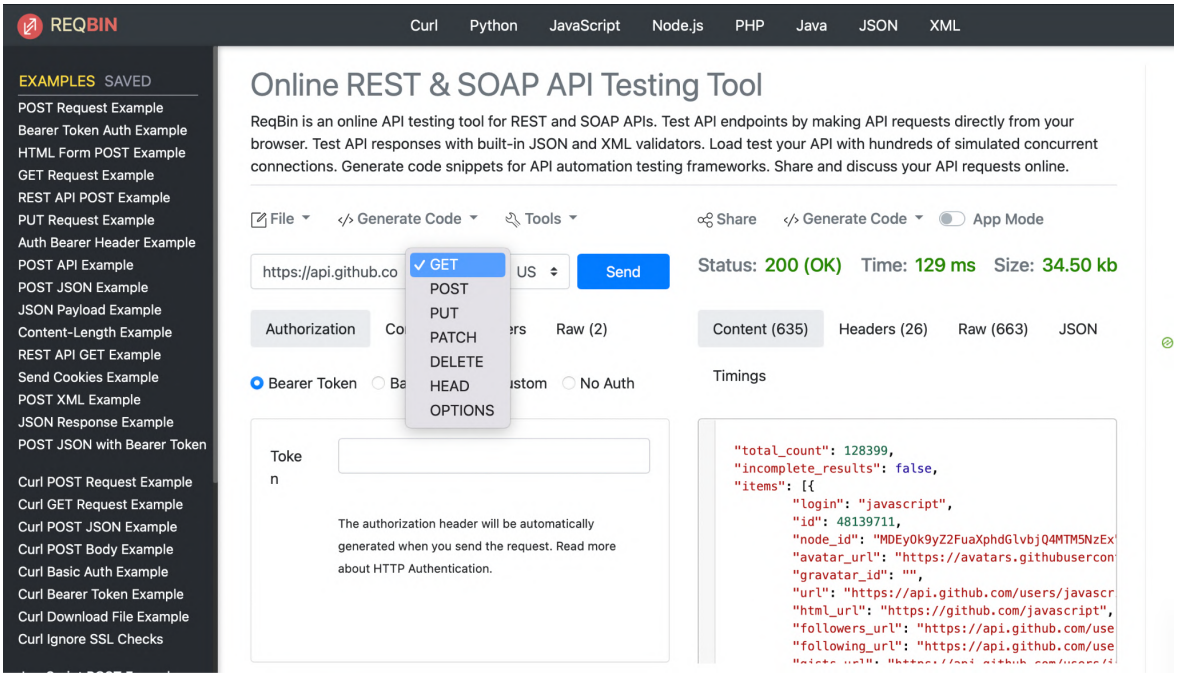
Name/Surname

Name	Link	Description	Price
Genderize.io	https://genderize.io	Instantly answers the question of how likely a certain name is to be male or female and shows the popularity of the name.	1000 names/day free
Agify.io	https://agify.io	Predicts the age of a person given their name	1000 names/day free
Nataonalize.io	https://nationalize.io	Predicts the nationality of a person given their name	1000 names/day free

Pastebin/Leaks

Name	Link	Description	Price
HaveIBeenPwned	https://haveibeenpwned.com/API/v3	allows the list of pwned accounts (email addresses and usernames)	$3.50 pe month
Psdmp.ws	https://psbdmp.ws/api	search in Pastebin	$9.95 pe 10000 requests
LeakPeek	https://psbdmp.ws/api	searc in leaks databases	$9.99 pe weeks unlimited access
BreachDirectory.com	https://breachdirectory.com/api?lang=en	search domain in data breaches databases	FREE
LeekLookup	https://leak-lookup.com/api	search domain, email_address, fullname, ip address, phone, password, username in leaks databases	10 reque FREE
BreachDirectory.org	https://rapidapi.com/rohan-patra/api/breachdirectory/pricing	search domain, email_address, fullname, ip address, phone, password, username in leaks	50 reque in month(FR

It is not necessary to write a separate Python script to test different APIs. It is better to use special online services that can simulate different types of requests and authorization methods.



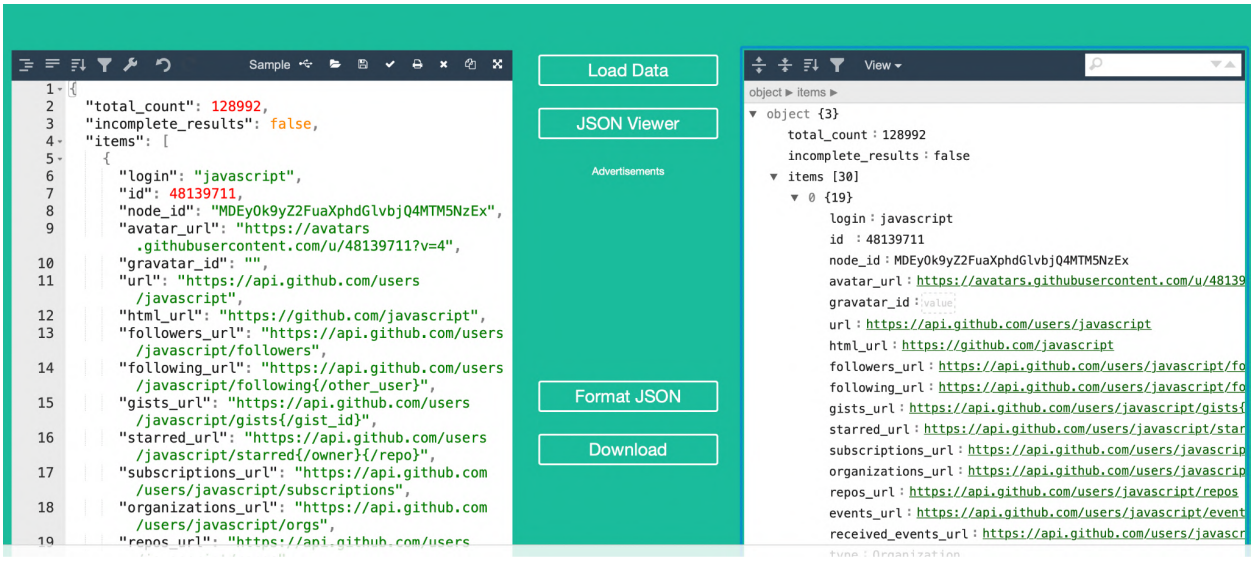
For example, reqbin.com or REST API Tester.

We will come back to the topic of network requests when we talk about JSON files, scraping and the use of proxy servers. We will learn how to add headers to the query and extract data from the response texts.

Day 6. JSON

In the last lesson, we talked about the fact that a lot of useful data for investigations can be obtained through various APIs. Many of them return data in JSON (JavaScript Object Notation) format (as well as CSV and XML, but we will talk about these formats in the next lessons).

In the last lesson, you saw a very good example of JSON data when working with the Github API (documentation).



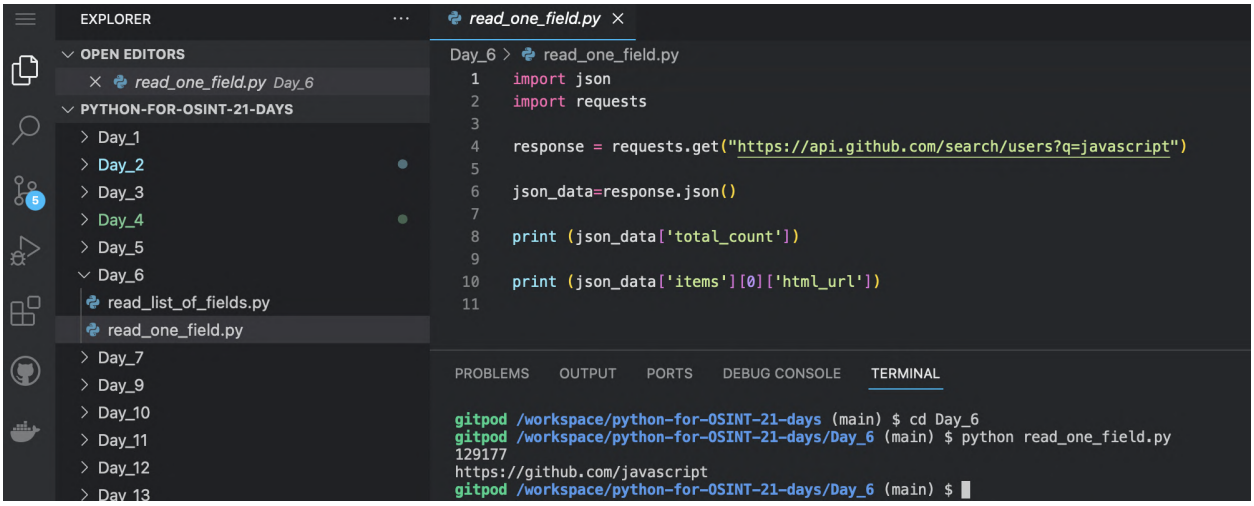
In response to the query, we get a list of 30 objects (items[0], items[1], items[2] etc), each corresponding to a specific user.

Each object has properties that store information about the user: login, html_url, id, followers_url etc.

Now let's try to extract data from JSON files using code. The JSON package (https://docs.python.org/3/library/json.html) is available in Python by default and does not require installation.

Reading one field

Run read_one_field.py:



Import the json and requests modules:

```
import json
import requests
```

Then make the same request to the Github API that we did in the previous lesson:

```
response =
requests.get("https://api.github.com/search/users?q=javascript")
```

Assign to the variable the value of the response to the query in json format:

```
Json_data = response.json()
```

Output the total number of results:

```
print (json_data['total_count'])
```

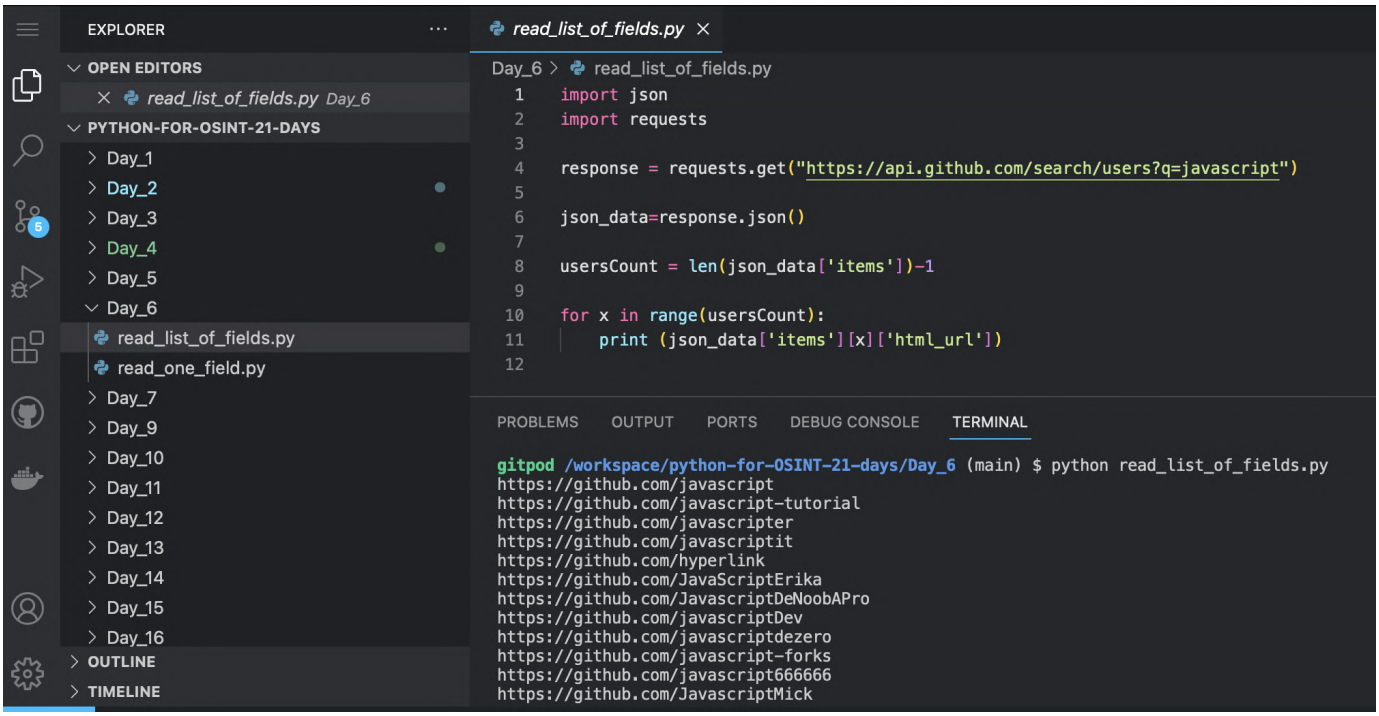
Output the link to the first Github profile from the results:

```
print (json_data['items'][0]['html_url'])
```

But most often we need to extract not a single value, but information about a whole list of objects. For example, the links to Github user profiles from the example above.

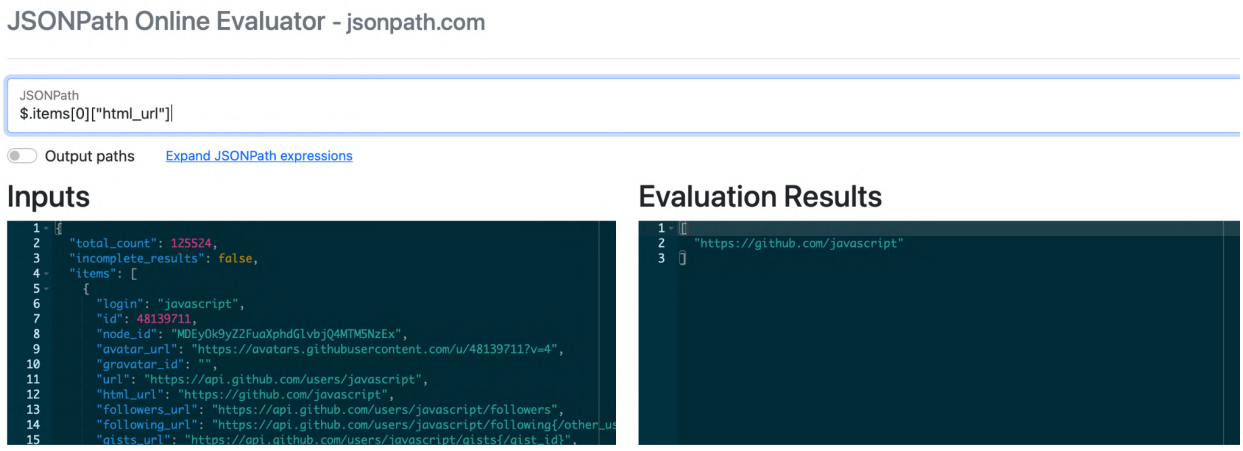
Reading list of fields

Run read_list_of_fields.py:



Import json and requests packages:

```
import json
import requests

# Make a request to API:

response =
requests.get("https://api.github.com/search/users?q=javascript")

# Get the result in JSON format:

json_data=response.json()

# Count the number of results:

usersCount = len(json_data['items'])-1

# Print a link to each result in loop:

for x in range(usersCount):
  print (json_data['items'][x]['html_url'])
```



It often happens that the structure of JSON files is quite complicated and it is difficult to understand how to mark the path to certain data. Special

services can help you figure this out. For example, https://jsonpath.com/ or https://jsonpathfinder.com.

And before you write any code to process JSON files, remember that sometimes it's easier to convert them to CSV files and just cut out the columns with the data you need:

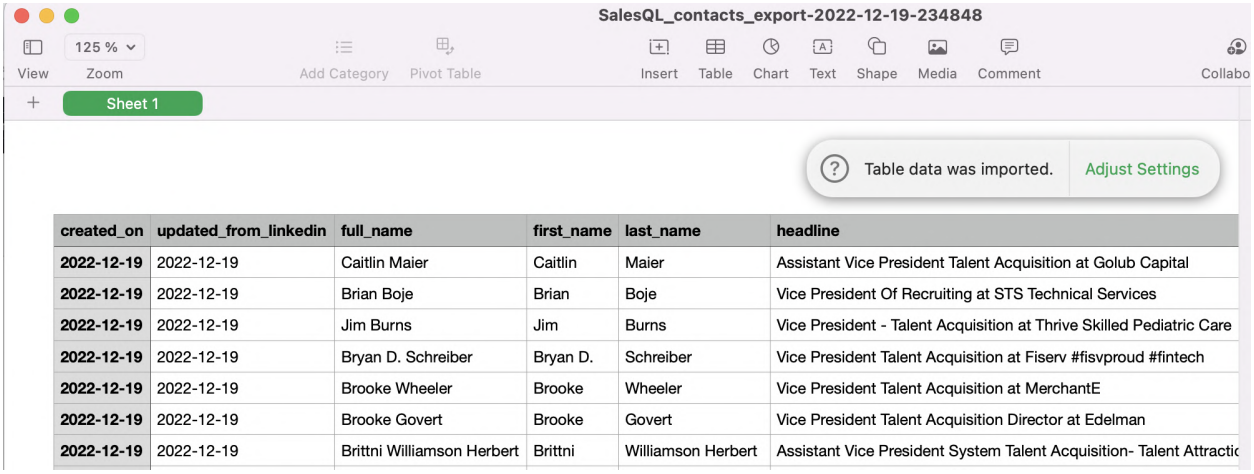
JSON to CSV online converter

Day 7. CSV

CSV (Comma-Separated Values) - is one of the most popular formats for storing tabular data.

Here is an example of how a CSV file looks when opened in a text editor.

And this is how it looks when you open Numbers (MS Excel equivalent for Mac).

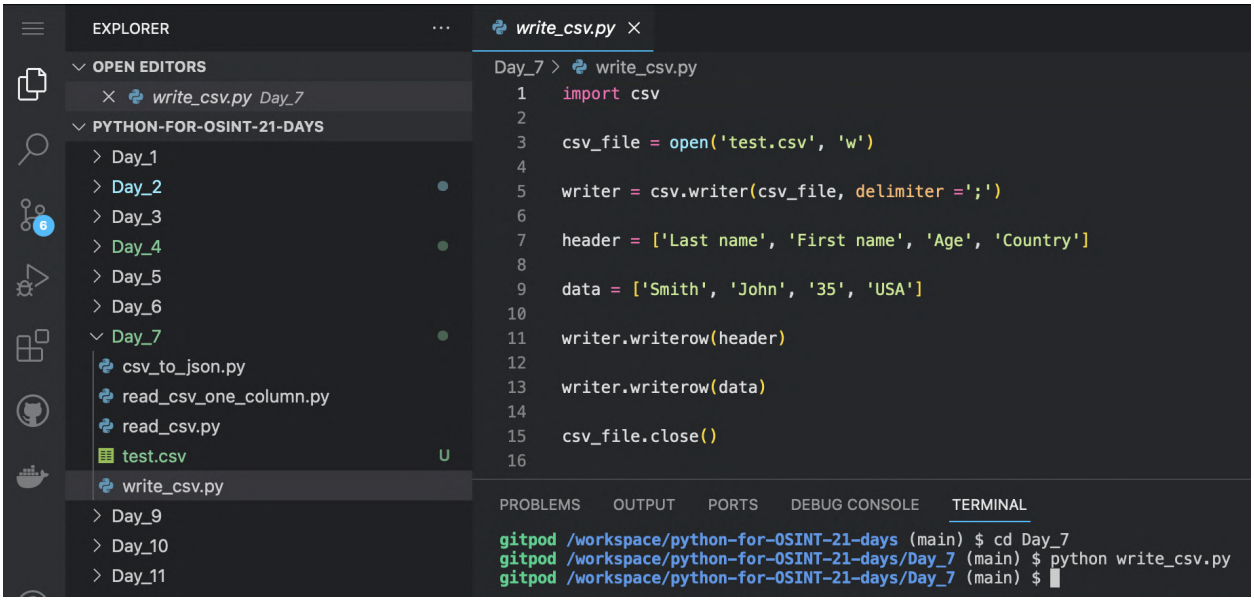


created_on	updated_from_linkedin	full_name	first_name	last_name	headline
2022-12-19	2022-12-19	Caitlin Maier	Caitlin	Maier	Assistant Vice President Talent Acquisition at Golub Capital
2022-12-19	2022-12-19	Brian Boje	Brian	Boje	Vice President Of Recruiting at STS Technical Services
2022-12-19	2022-12-19	Jim Burns	Jim	Burns	Vice President - Talent Acquisition at Thrive Skilled Pediatric Care
2022-12-19	2022-12-19	Bryan D. Schreiber	Bryan D.	Schreiber	Vice President Talent Acquisition at Fiserv #fisvproud #fintech
2022-12-19	2022-12-19	Brooke Wheeler	Brooke	Wheeler	Vice President Talent Acquisition at MerchantE
2022-12-19	2022-12-19	Brooke Govert	Brooke	Govert	Vice President Talent Acquisition Director at Edelman
2022-12-19	2022-12-19	Brittni Williamson Herbert	Brittni	Williamson Herbert	Assistant Vice President System Talent Acquisition- Talent Attracti

Let's try to create a CSV file using the CSV package (https://docs.python.org/3/library/csv.html).

Writing CSV file

Run write_csv.py:



Import the CSV package (it is available by default):

```
import csv
```

Open and create a test.csv file:

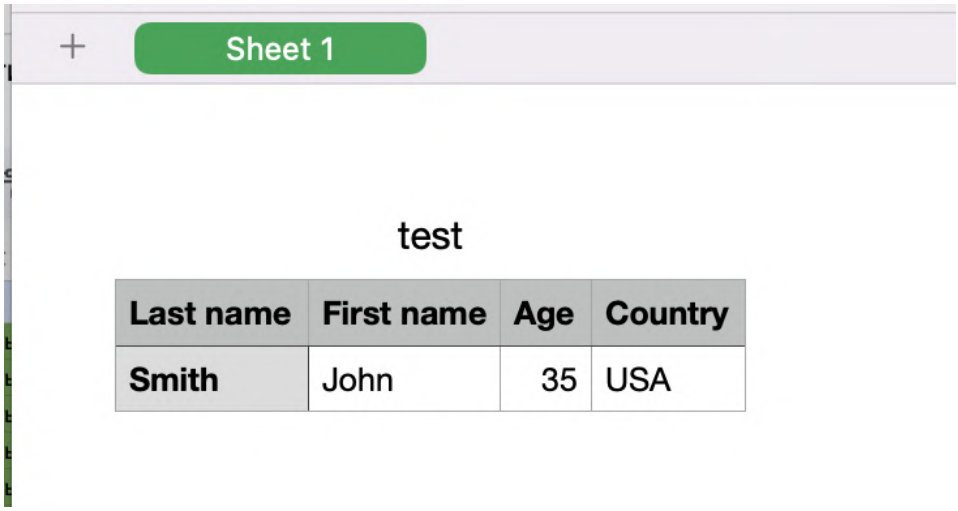
```
csv_file = open('test.csv', 'w')
```

Create csv writer object:

```
writer = csv.writer(csv_file, delimiter =';')
```

```
# Create a list with data headers:

header = ['Last name', 'First name', 'Age', 'Country']

# Create a list with one line of data:

data = ['Smith', 'John', '35', 'USA']

# Write the data headers to the file:

writer.writerow(header)

# Write a line of data to the file:

writer.writerow(data)

# Close the test.csv file:

csv_file.close()
```

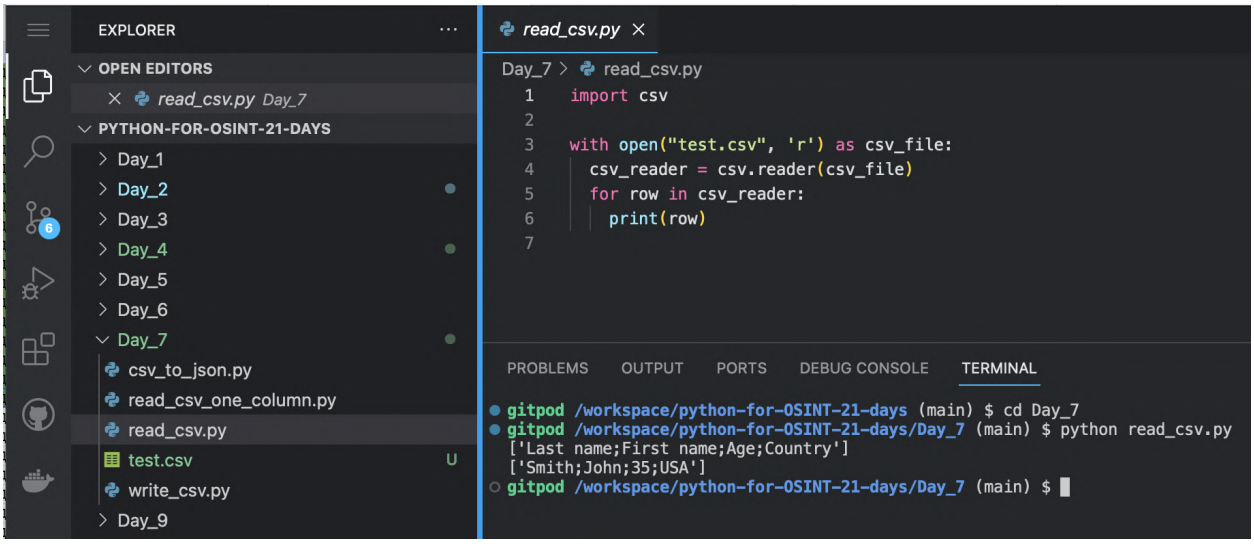
The CSV file created in this way can be opened in any spreadsheet editor: Excel, Numbers, Google Sheet etc.



Now, let's try to read the contents of the CSV file.

Reading CSV file

Run read_csv.py:



Import the csv package:

```
import csv
```

Open file test.csv:

```
with open("test.csv", 'r') as csv_file:
```

Create csv.reader object:

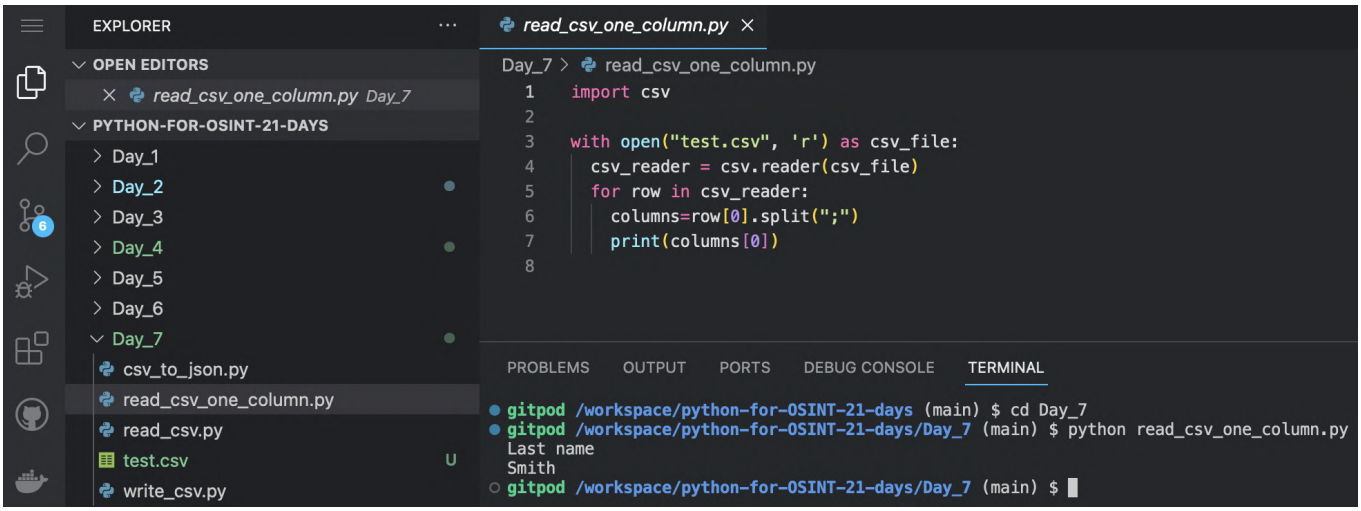
```
csv_reader = csv.reader(csv_file)
```

Print lines one by one:

```
for row in csv_reader:
  print(row)
```

Now let's try reading the data from a separate column.

Run read_csv_one_column.py:



Import the csv package:

```
import csv
```

Open the test.csv file:

```
with open("test.csv", 'r') as csv_file:
```

Create csv reader object:

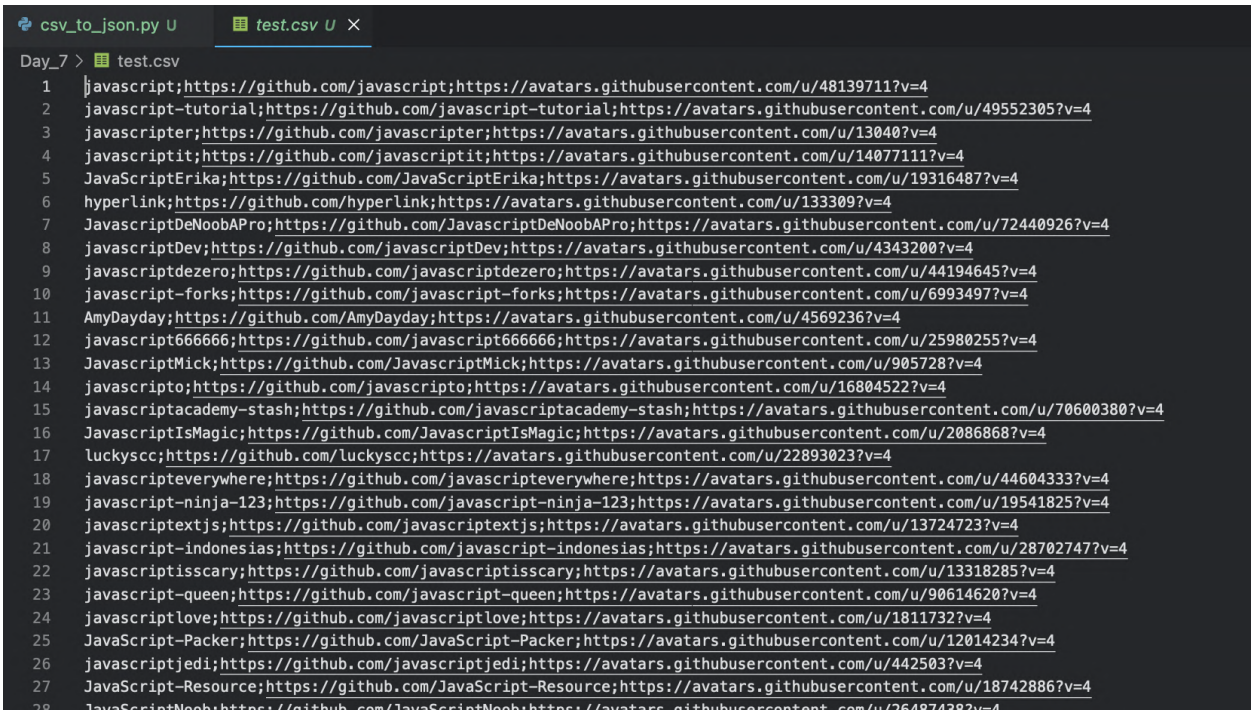
```
csv_reader = csv.reader(csv_file)
```

One by one divide the string into columns, using delimiter - semicolon:

```
for row in csv_reader:
  columns=row[0].split(";")
```

And print first column:

```
print(columns[0])
```

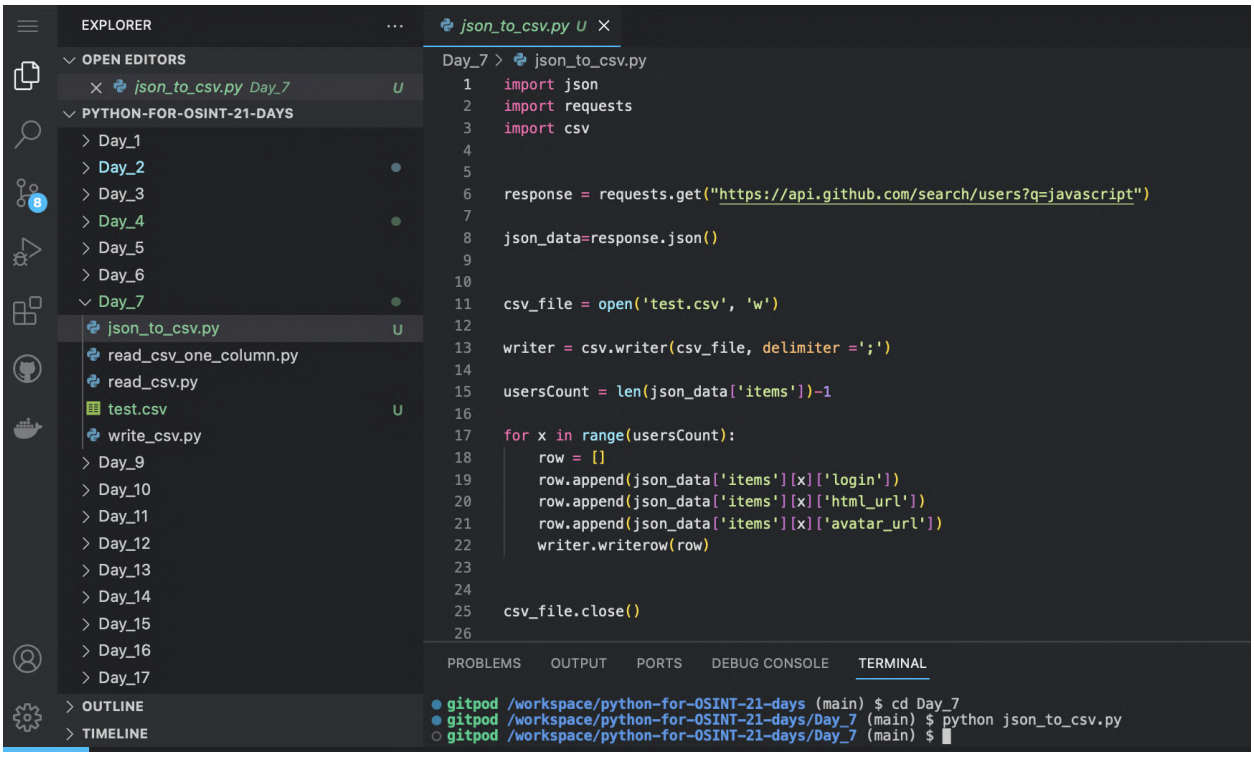


JSON to CSV

Sometimes you need to convert data from JSON to CSV so that it can be conveniently viewed and opened in Microsoft Excel/Google Sheet.

You can do it with special services like csvjson.com **(and that would be the best solution)**.

But I will show you how to do it with Python code to reinforce what you have learned in the last two days.



Run json_to_csv.py:

Importing json, csv and requests packages:

```
import json
import requests
import csv
```

Make a request to Github API:

```
response =
requests.get("https://api.github.com/search/users?q=javascript")
```

Get data in JSON format:

```
json_data=response.json()
```

Open and simultaneously create file test.csv:

```
csv_file = open('test.csv', 'w')
```

Create csv_writer object:

```
writer = csv.writer(csv_file, delimiter =';')
```

Count the number of found users:

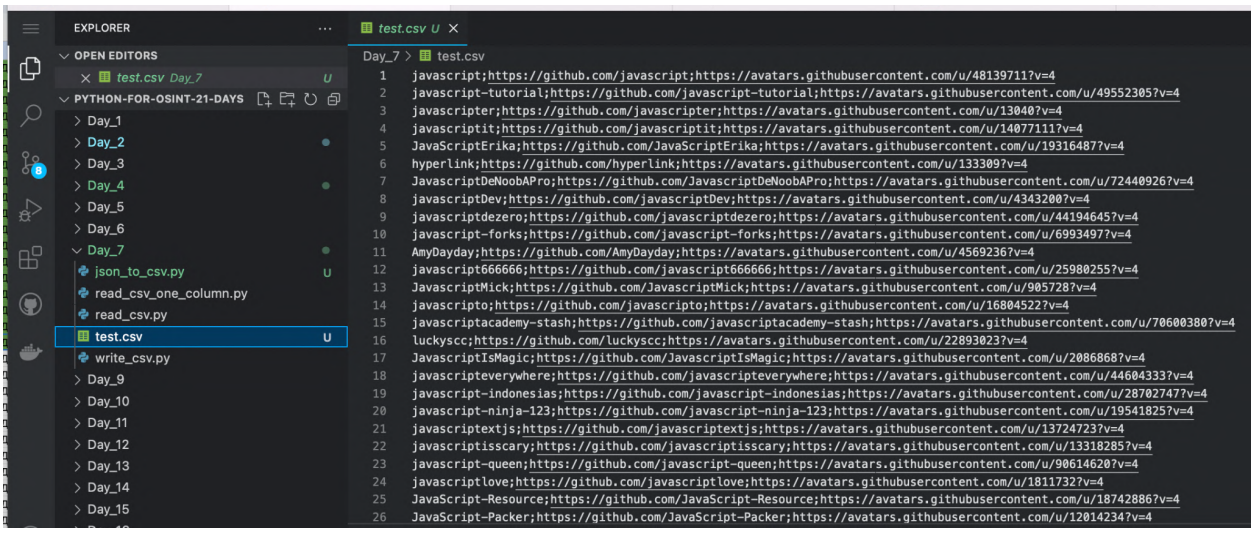
```
usersCount = len(json_data['items'])-1
```

Pass each line of JSON data one by one, create empty string object, add login, link to profile and link to avatar, write string to csv file:

```
for x in range(usersCount):
  row = []
  row.append(json_data['items'][x]['login'])
  row.append(json_data['items'][x]['html_url'])
  row.append(json_data['items'][x]['avatar_url'])
  writer.writerow(row)
```

Close test.csv file:

```
csv_file.close()
```



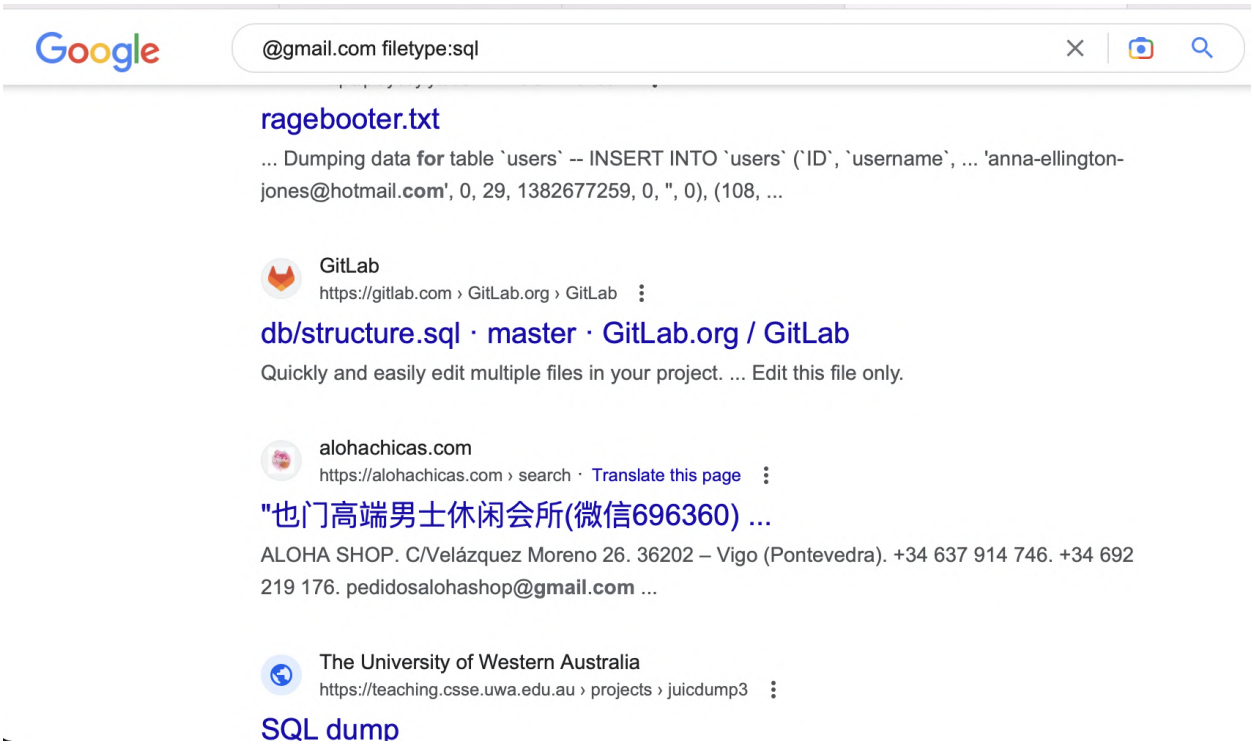
This is what the contents of the test.csv file should look like after running the csv_to_json.py script.

Day 8. Databases

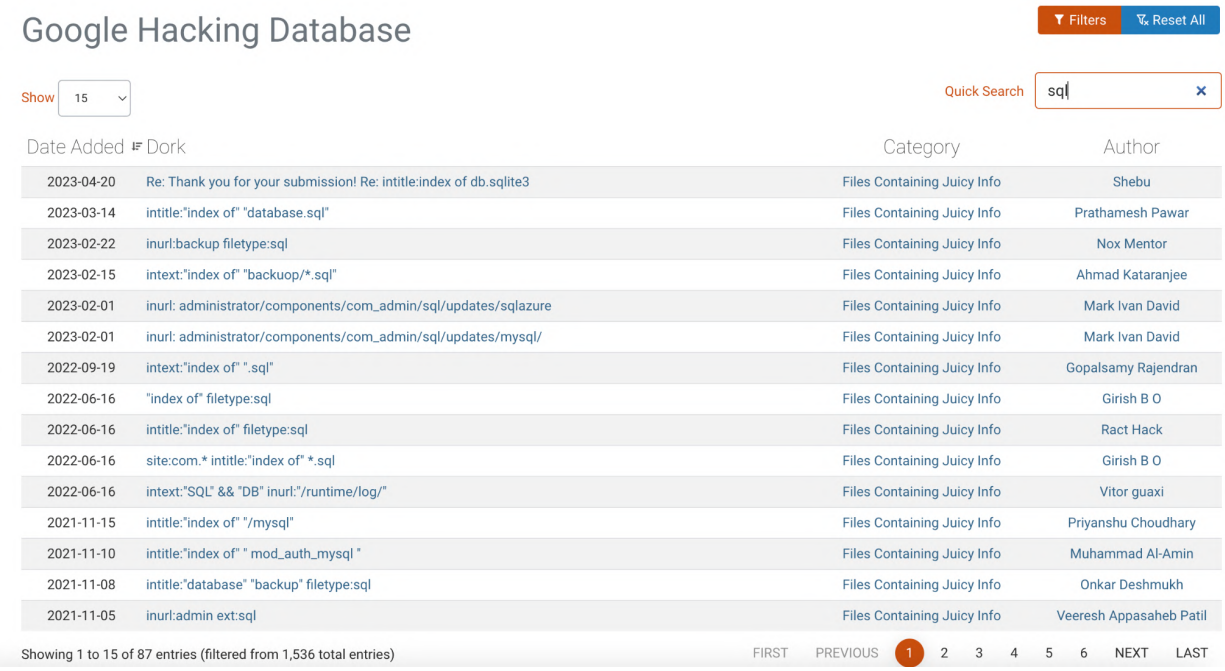
There are a lot of python packages to work with almost all popular databases: MySQL, PostgreSQL, MongoDB, Redis, Elasticsearch etc. I decided in this course not to go over code examples for any one database, but just to give some universal advice.

SQL (Structured Query Language) format files are very often encountered in investigations. This format stores database dumps, in which you can often find useful contact information (a common example is a list of emails and phone numbers of employees of the company).

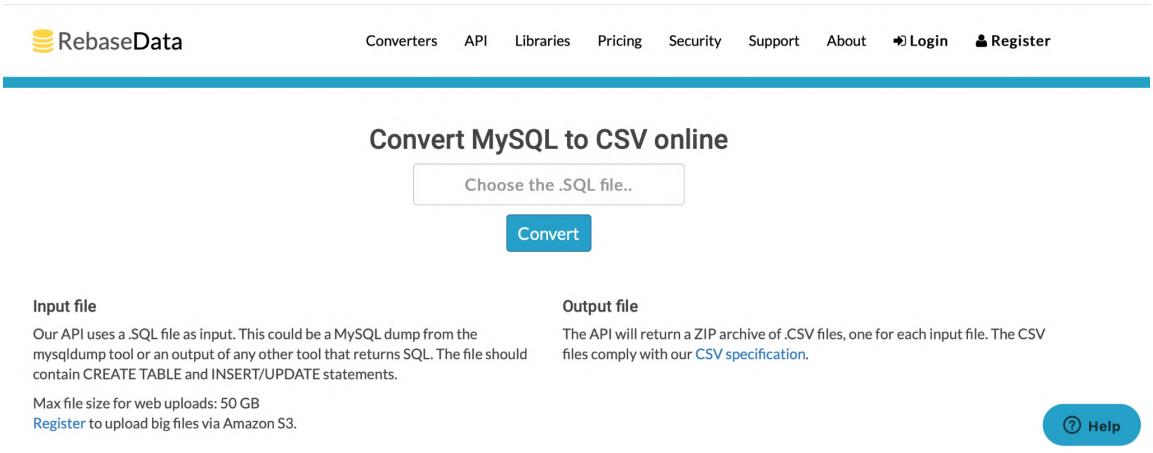
For example, they can sometimes end up in Google searches.



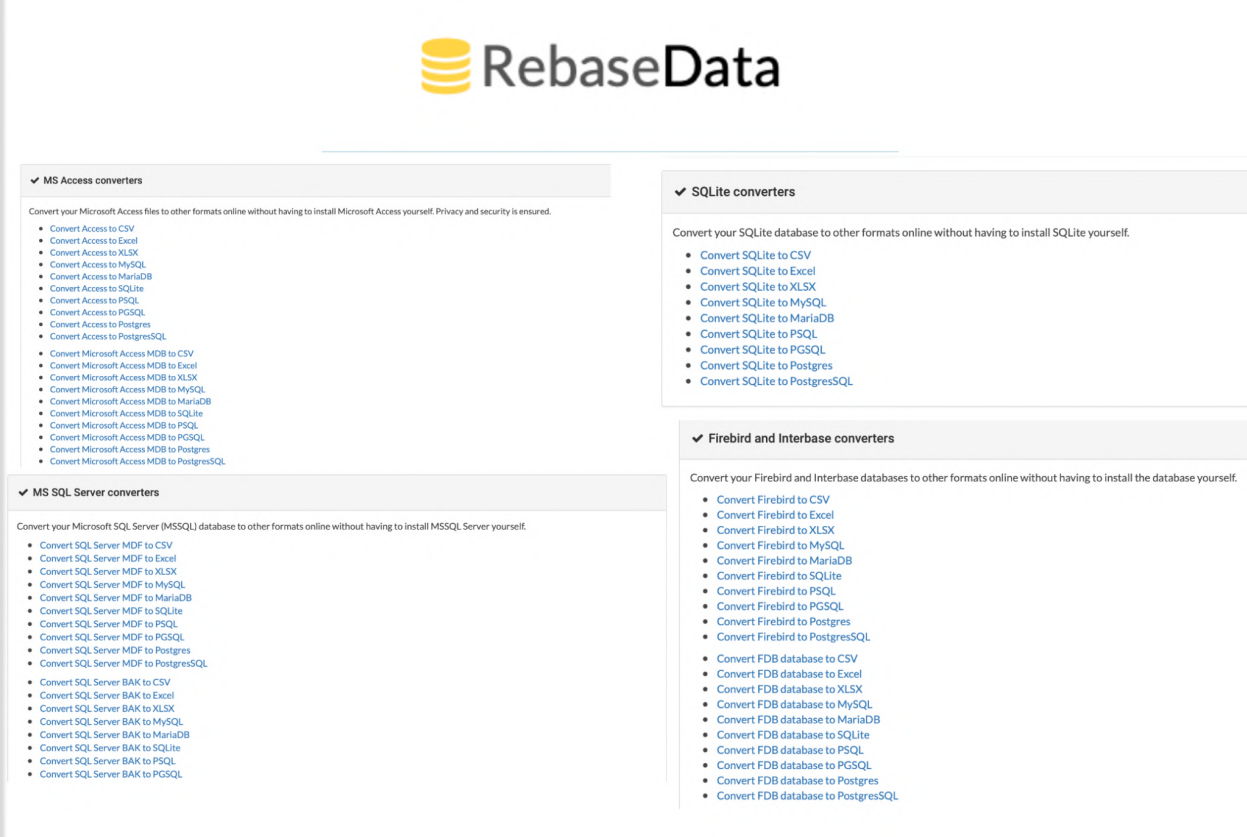
If you do a Google Hacking Database search for "sql" (particularly in Juicy Info Dorks section), you will find over 1,500 example queries to find data in .sql files.



The easiest way to view such a file is simply to convert it to CSV and open it in Excel/Numbers/Google Sheet. This free online converter will help you. For example, Rebsase SQL to CSV.



Also useful for investigation contact information that can be stored and databases of other formats: MS Access (.MDB), SQl Server (.MDF), SQLite (.sqlite, .sqlite3, .db, .db3, .s3db, .sl3), Firebird (.FBD) and many others.



They can also be converted to CSV using online converters:

Rebasedata.com

Anyconv.com

101convert.com

In this lesson, we will not run sample code. As practice, I'll just recommend that you find database files with data contacts on Google, see how they're

arranged, and convert them to CSV. Use the filetype:pdf operator and sample queries from Google Hacking Database.

Day 9. Automate the collection of search results

There are a huge number of Python tools for collecting search results from different search engines. Many of them are designed to search for vulnerable sites and juicy info (for example, tables with personal contact data) using Google Dorks.

They save a huge amount of time looking at search results. Because they can automatically analyze the content of found web pages.

Here are a few examples:

Email Finder (https://github.com/Josue87/EmailFinder) - search emails from a domain with Google, Bing, Baidu.
StartPageParser (https://github.com/knassar702/startpage-parser) - collect search results from startpage search engine (based on google.com results). This allows you to collect Google search results without having to think about getting banned by Google.
Searcher (https://github.com/davemolk/searcher) - collect search results from Ask, Bing, Brave, Duck Duck Go, Yahoo, and Yandex.
DDGR (https://github.com/jarun/ddgr) - collect DuckDuckGo search results.
Search Engines Scraper (https://github.com/tasos-py/Search-Engines-Scraper) - collect search results from 11 search engines.

Today we learn to use duckduckgo-search package https://pypi.org/project/duckduckgo-search/. In my opinion, this is one of the best options in terms of the combination of "ease of use" + "power of functionality".

```
gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_9
gitpod /workspace/python-for-OSINT-21-days/Day_9 (main) $ pip install duckduckgo_search
Collecting duckduckgo_search
  Downloading duckduckgo_search-2.9.2-py3-none-any.whl (30 kB)
Requirement already satisfied: click>=8.1.3 in /workspace/.pyenv_mirror/user/current/lib/python3.11/site-packages (from duckduckgo_search) (8.1.3)
Collecting diskcache>=5.6.1 (from duckduckgo_search)
  Downloading diskcache-5.6.1-py3-none-any.whl (45 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 45.6/45.6 kB 2.1 MB/s eta 0:00:00
Collecting requests>=2.29.0 (from duckduckgo_search)
  Downloading requests-2.29.0-py3-none-any.whl (62 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.5/62.5 kB 3.5 MB/s eta 0:00:00
Requirement already satisfied: charset-normalizer<4,>=2 in /workspace/.pyenv_mirror/user/current/lib/python3.11/site-packages (from requests>=2.29.0->duc
kduckgo_search) (2.1.1)
Requirement already satisfied: idna<4,>=2.5 in /home/gitpod/.pyenv/versions/3.11.1/lib/python3.11/site-packages (from requests>=2.29.0->duckduckgo_search
) (3.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /home/gitpod/.pyenv/versions/3.11.1/lib/python3.11/site-packages (from requests>=2.29.0->duckduck
go_search) (1.26.15)
Requirement already satisfied: certifi>=2017.4.17 in /home/gitpod/.pyenv/versions/3.11.1/lib/python3.11/site-packages (from requests>=2.29.0->duckduckgo_
search) (2022.12.7)
Installing collected packages: requests, diskcache, duckduckgo_search
  Attempting uninstall: requests
    Found existing installation: requests 2.28.1
    Uninstalling requests-2.28.1:
      Successfully uninstalled requests-2.28.1
Successfully installed diskcache-5.6.1 duckduckgo_search-2.9.2 requests-2.29.0
gitpod /workspace/python-for-OSINT-21-days/Day_9 (main) $
```

Install package from pip:

```
pip install duckduckgo_search
```

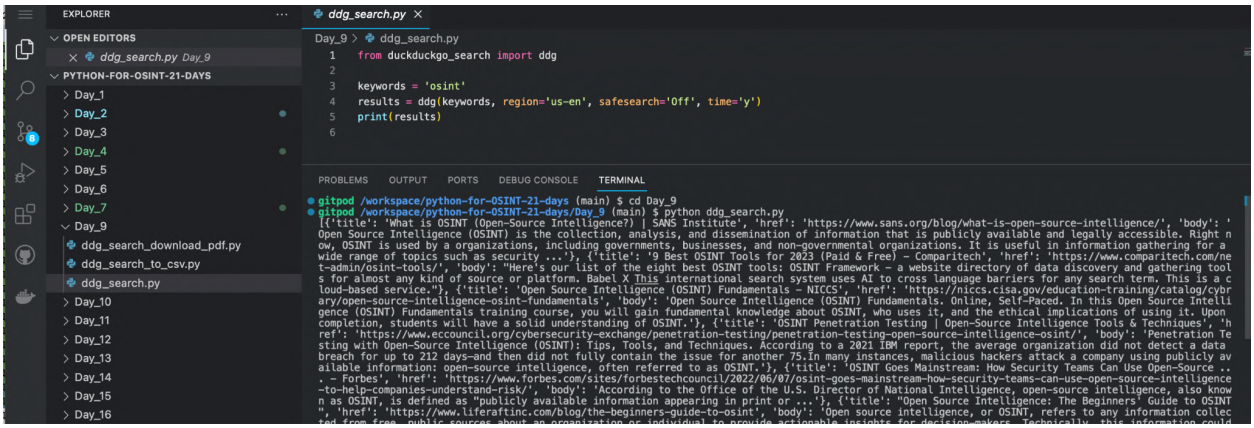
```
gitpod /workspace/python-for-OSINT-21-days (main) $ python -m duckduckgo_search --help
Usage: duckduckgo_search [OPTIONS] COMMAND1 [ARGS]... [COMMAND2 [ARGS]...]...

Options:
  --help  Show this message and exit.

Commands:
  answers
  images
  maps
  news
  suggestions
  text
  translate
  version
  videos
gitpod /workspace/python-for-OSINT-21-days (main) $
```

Check installation:

```
python -m duckduckgo_search --help
```

Please remember this flag. It works for most Python packages. If you type --help or -h after the command name, help information about its use will be displayed.

Run ddg_search.py:



Importing the ddg package:

```
from duckduckgo_search import ddg
```

Create a variable with a search request:

```
keywords = 'osint'
```

Send a search request, specifying that we want to see the search results for the US with safe search turned off:

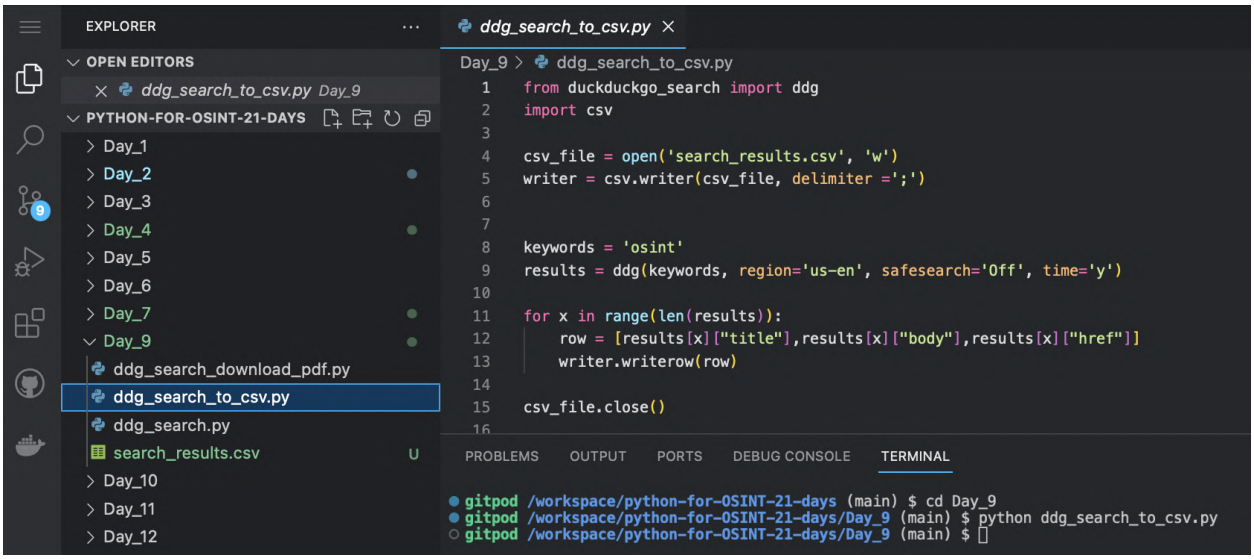
```
results = ddg(keywords, region='us-en', safesearch='Off', time='y')
```

Print the results:

```
print(results)
```

Simply displaying the results on the screen is not very useful. The same could be done in the browser. Let's try to save them to a CSV file so that they can be automatically analyzed later.

Run ddg_search_to_csv.py:



Importing ddg and csv modules:

```
from duckduckgo_search import ddg
import csv
```

Open the file search_results.csv:

```
csv_file = open('search_results.csv', 'w')
```

Create csv.writer object:

```
writer = csv.writer(csv_file, delimiter =';')
```

Create a variable with search queries:

```
keywords = 'osint'
```

Send a search request, specifying that we want to see the search results for the US with safe search turned off:

```
results = ddg(keywords, region='us-en', safesearch='Off', time='y')
```

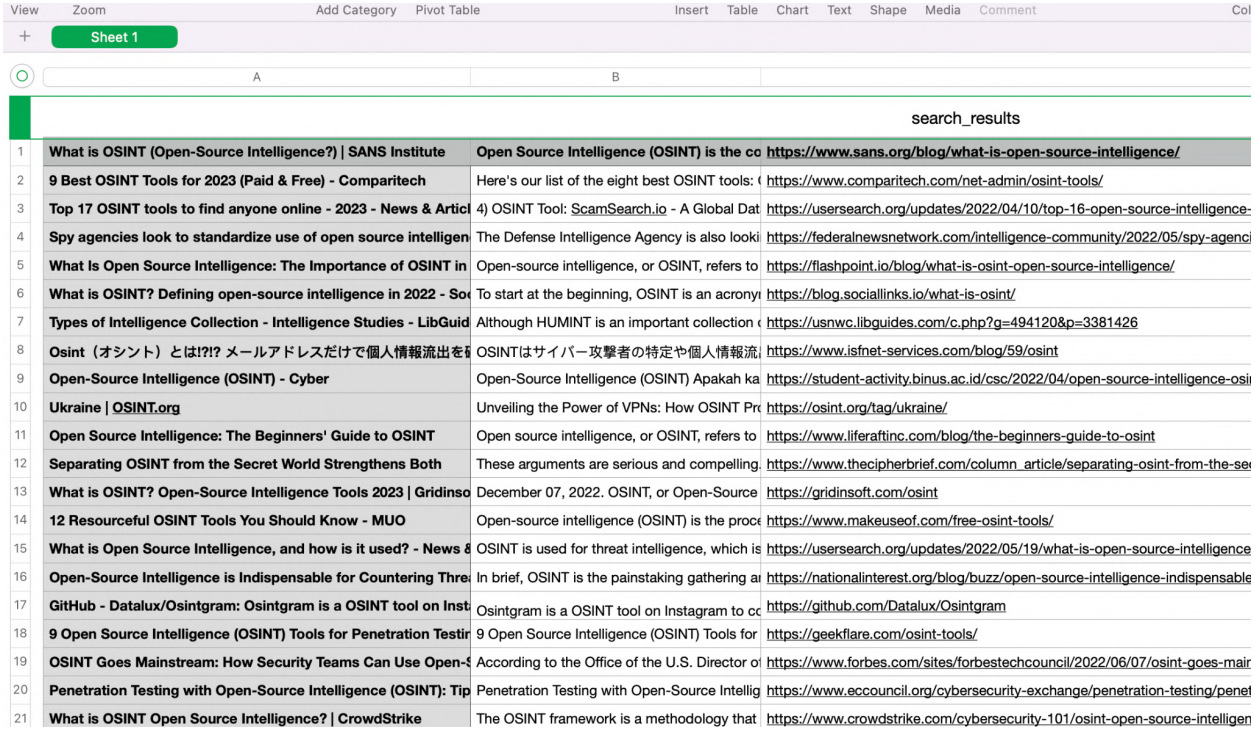
Go through the search results one by one and write each one in a line of the CSV file with three fields - title,body,href (note that each time you write a string, it creates a list with three elements):

```
for x in range(len(results)):
    row = [results[x]["title"],results[x]["body"],results[x]["href"]]
    writer.writerow(row)
```

Close search_results.csv file:

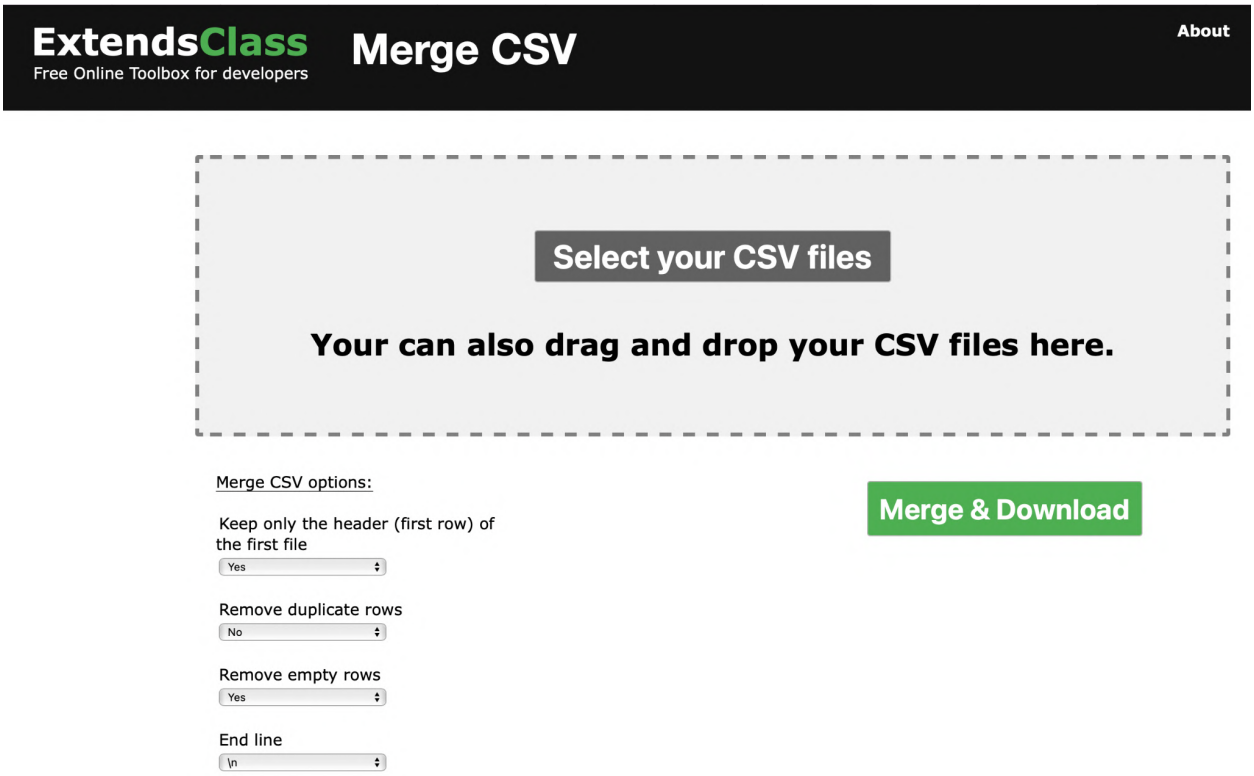
```
csv_file.close()
```

This is what the file search_results.csv looks like in Numbers:



	A	B	search_results
1	What is OSINT (Open-Source Intelligence?) | SANS Institute	Open Source Intelligence (OSINT) is the co	https://www.sans.org/blog/what-is-open-source-intelligence/
2	9 Best OSINT Tools for 2023 (Paid & Free) - Comparitech	Here's our list of the eight best OSINT tools: (	https://www.comparitech.com/net-admin/osint-tools/
3	Top 17 OSINT tools to find anyone online - 2023 - News & Artic	4) OSINT Tool: ScamSearch.io - A Global Dat	https://usersearch.org/updates/2022/04/10/top-16-open-source-intelligence-
4	Spy agencies look to standardize use of open source intelligen	The Defense Intelligence Agency is also looki	https://federalnewsnetwork.com/intelligence-community/2022/05/spy-agenci
5	What Is Open Source Intelligence: The Importance of OSINT in	Open-source intelligence, or OSINT, refers to	https://flashpoint.io/blog/what-is-osint-open-source-intelligence/
6	What is OSINT? Defining open-source intelligence in 2022 - Soc	To start at the beginning, OSINT is an acronyı	https://blog.sociallinks.io/what-is-osint/
7	Types of Intelligence Collection - Intelligence Studies - LibGuid	Although HUMINT is an important collection (	https://usnwc.libguides.com/c.php?g=494120&p=3381426
8	Osint（オシント）とは!?!? メールアドレスだけで個人情報流出を	OSINTはサイバー攻撃者の特定や個人情報流(	https://www.isfnet-services.com/blog/59/osint
9	Open-Source Intelligence (OSINT) - Cyber	Open-Source Intelligence (OSINT) Apakah ka	https://student-activity.binus.ac.id/csc/2022/04/open-source-intelligence-osi
10	Ukraine | OSINT.org	Unveiling the Power of VPNs: How OSINT Pro	https://osint.org/tag/ukraine/
11	Open Source Intelligence: The Beginners' Guide to OSINT	Open source intelligence, or OSINT, refers to	https://www.liferaftinc.com/blog/the-beginners-guide-to-osint
12	Separating OSINT from the Secret World Strengthens Both	These arguments are serious and compelling.	https://www.thecipherbrief.com/column_article/separating-osint-from-the-se
13	What is OSINT? Open-Source Intelligence Tools 2023 | Gridinso	December 07, 2022. OSINT, or Open-Source	https://gridinsoft.com/osint
14	12 Resourceful OSINT Tools You Should Know - MUO	Open-source intelligence (OSINT) is the proce	https://www.makeuseof.com/free-osint-tools/
15	What is Open Source Intelligence, and how is it used? - News &	OSINT is used for threat intelligence, which is	https://usersearch.org/updates/2022/05/19/what-is-open-source-intelligence
16	Open-Source Intelligence is Indispensable for Countering Threa	In brief, OSINT is the painstaking gathering ar	https://nationalinterest.org/blog/buzz/open-source-intelligence-indispensable
17	GitHub - Datalux/Osintgram: Osintgram is a OSINT tool on Inst	Osintgram is a OSINT tool on Instagram to co	https://github.com/Datalux/Osintgram
18	9 Open Source Intelligence (OSINT) Tools for Penetration Testir	9 Open Source Intelligence (OSINT) Tools for	https://geekflare.com/osint-tools/
19	OSINT Goes Mainstream: How Security Teams Can Use Open-S	According to the Office of the U.S. Director of	https://www.forbes.com/sites/forbestechcouncil/2022/06/07/osint-goes-mair
20	Penetration Testing with Open-Source Intelligence (OSINT): Tip	Penetration Testing with Open-Source Intellig	https://www.eccouncil.org/cybersecurity-exchange/penetration-testing/penet
21	What is OSINT Open Source Intelligence? | CrowdStrike	The OSINT framework is a methodology that	https://www.crowdstrike.com/cybersecurity-101/osint-open-source-intelligen

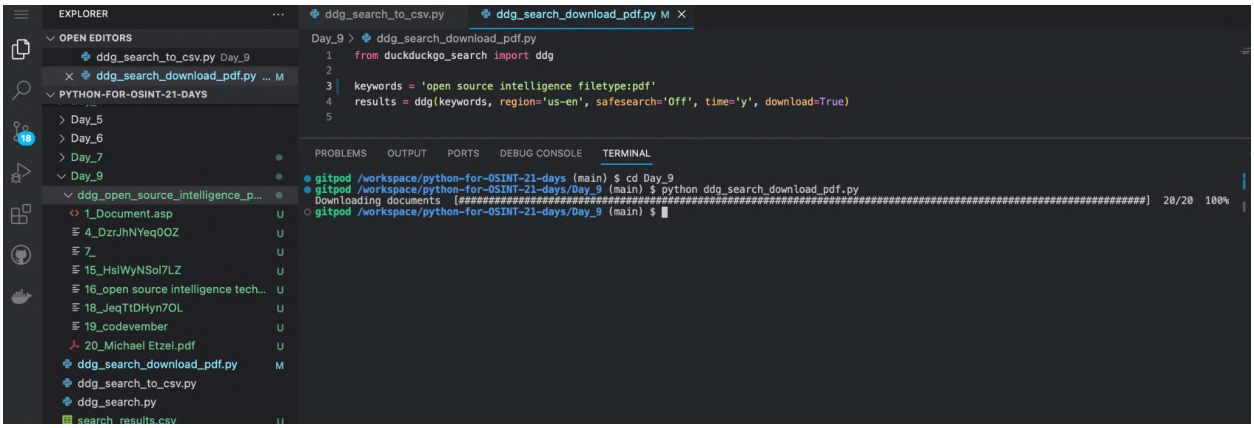
Python makes it easy to find and manipulate files in folders (we'll talk more about that in Lesson 14), but sometimes it's more convenient just to combine several CSV files into one to save your time while writing code.



You can do this with any service you can find in Google by searching for "Merge CSV files online". For example, https://extendsclass.com/merge-csv.html

This package also allows you to download page content from search results. You can download all found files (html, pdf, xlsx etc) and then automatically analyze them or just do a simple keyword search in them.

Run ddg_search_download_pdf.py:



Importing the ddg package:

```
from duckduckgo_search import ddg
```

Create a variable with search queries (include filetype:pdf):

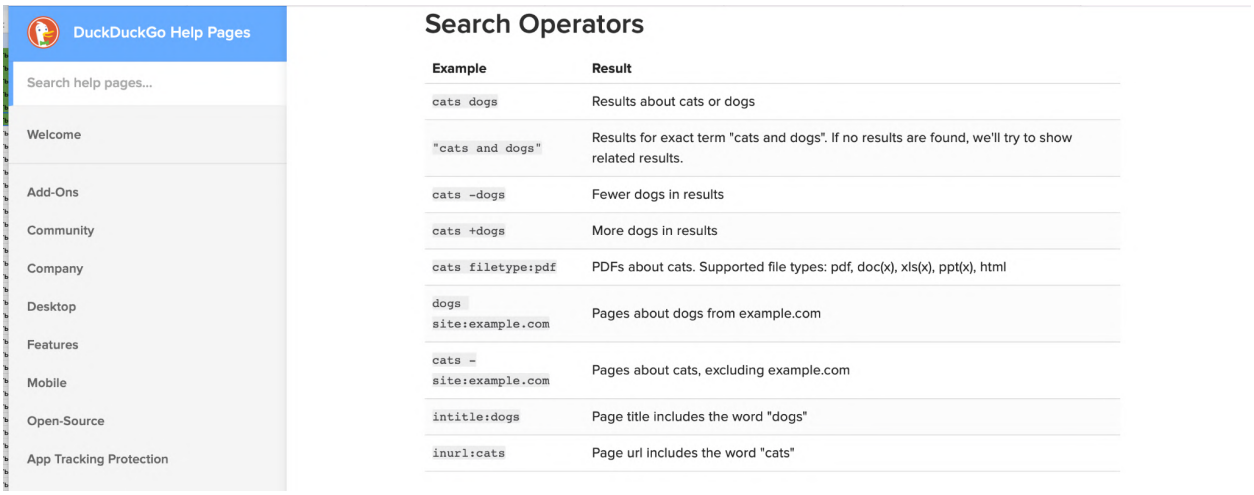
```
keywords = 'open source intelligence filetype:pdf'
```

Send a search request, specifying that we want to see the search results for the US with download option turned on:

```
results = ddg(keywords, region='us-en', safesearch='Off', time='y', download=True)
```

To change the number of downloadable files, set the max_results parameter in ddg().

With duckduckgo_search packages you can you can also collect Answers, Images, Videos, News, Maps, Suggestions, and translate text.



Note that in the last example we used the extended search operator filetype:pdf. Other advanced search operators can also be used in queries for duckduckgo_search.

[List of DuckDuckGo Search Operators]

It is worth noting that the ability to use advanced search operators at every opportunity is a very useful skill for every OSINT specialist. Here is a list of reference articles with advanced search operators for search engines, social media platforms, mailboxes, and other services:

[Advanced search operators list]

Day 10. Scraping

Scraping is the extraction of data from website.

The most important thing to know about scraping and Python is that writing your own script from scratch for each task is most often not the best solution. It is better to try different ready-made tools first.

Web Scraper Chrome Extension (https://chrome.google.com/webstore/detail/web-scraper-free-web-scra/jnhgnonknehpejjnehehllkliplmbmhn)

As I write this course, the world is changing rapidly and AI scraping is actively evolving.

Browse AI
https://www.browse.ai
AnyPicker https://chrome.google.com/webstore/detail/anypicker-ai-powered-no-c/bjkpgfhekfmdffdphnniobddhkjlmmlj
ScrapeStorm
https://www.scrapestorm.com

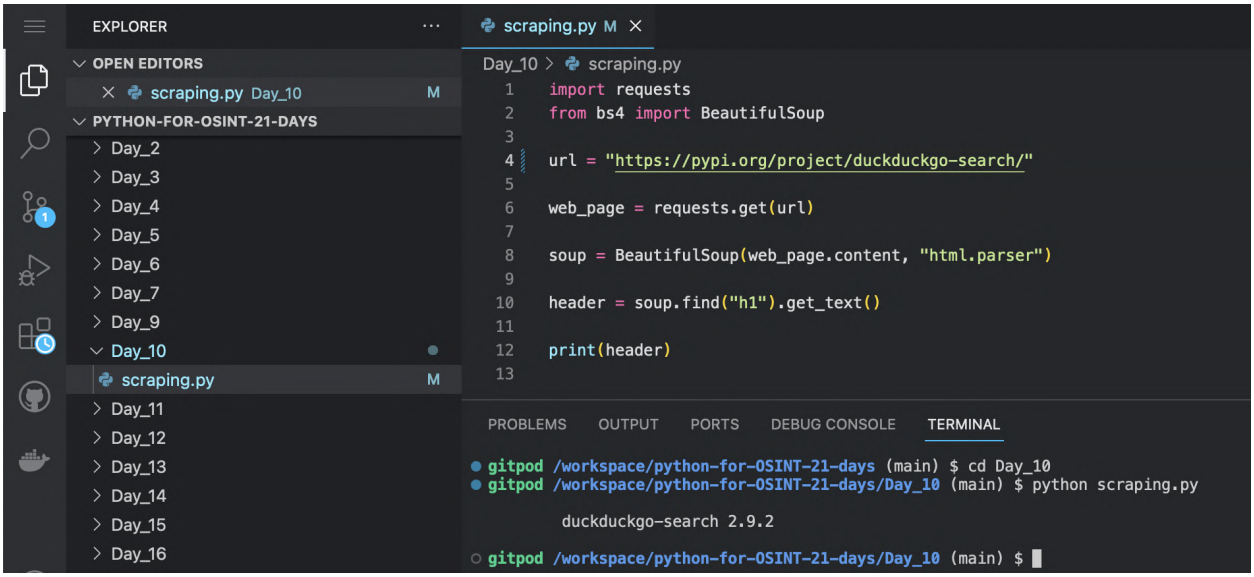
And almost every day some new AI scraping tool appears.

If you need to collect data from any popular social networks, try to find any solution made specifically for a particular platform. For example, YouTube tool (https://github.com/nlitsme/youtube_tool) for YouTube or Stweet (https://github.com/markowanga/stweet) for Twitter.

But sometimes you may encounter a problem for which there are no ready-made solutions and you want to write your own script to solve it. There are a lot of Python packages for scraping: Scrapy, Selenium, ZenRows etc.

We will use BeautifulSoup package (https://pypi.org/project/beautifulsoup4/) for scraping. It is installed by default.

Beautifulsoup package can also be useful to work with data in XML format (you may encounter it in particular when retrieving data from some API). In this case, in addition to Beautifulsoup you should use LXML package (https://lxml.de).



Run scraping.py:

Import requests and BeautifulSoup packages:

```
import requests
from bs4 import BeautifulSoup
```

Create variable with URL of web page:

```
url = "https://pypi.org/project/duckduckgo-search/"
```

```
# Make https request:

web_page = requests.get(url)

# Create html.parser object:

soup = BeautifulSoup(web_page.content, "html.parser")

# Find h1 header in html code of the web page:

header = soup.find("h1").get_text()

# Print h1 header:

print(header)
```

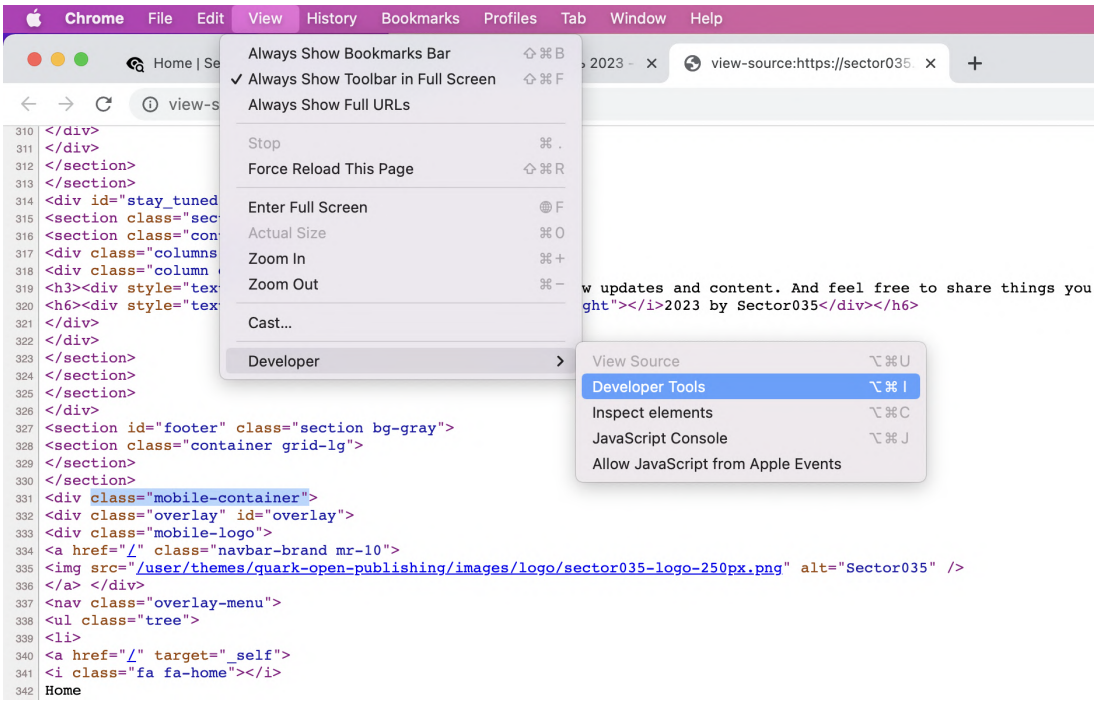
We use CSS-selectors to find elements on a web page:

“h1” - element with h1 tag.
“.headers” - elements with headers class.
“#header” - elements with header id.
“div.reblocks” - elements with div tag and redblocks class.
“[autofocus="true"]”- elements with the autofocus attribute with a value of “true”

More CSS-selectors:

https://www.freecodecamp.org/news/css-selectors-cheat-sheet/

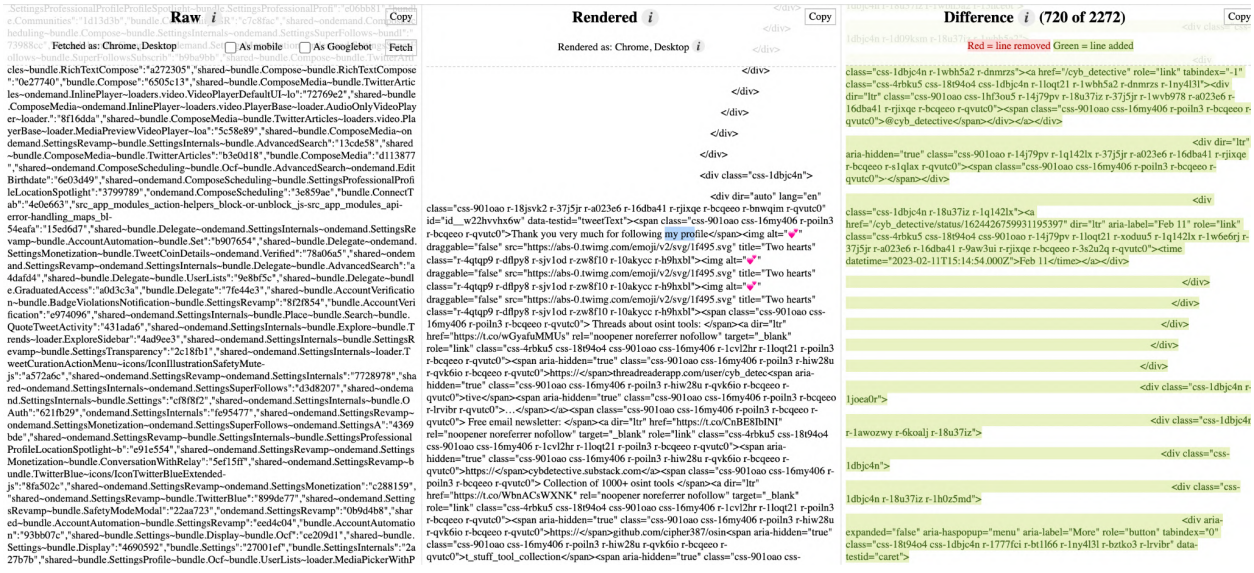
The easiest way to find out which selector stands for a certain html-element is to look at the source code of the page with the help of developer tools, which are available in every popular browser.



And for scraping pages with a complex structure (which contains many nested elements), you can use special browser extensions that display the full "path" to the element. For example, [HTML DOM Navigation]

It often happens that the code that is displayed when the site is loaded with Python scripts is very different from what is displayed in the browser. This is caused by the fact that some elements are added after the page has been loaded by executing JavaScript code.

To see what I mean, try opening some Twitter account code in the View Rendered Source extension.



With it you can visually compare how the HTML code looks immediately after receiving a request from the server, and how it looks after performing certain actions on the page (try scrolling down the ribbon a bit and restarting the extension again).

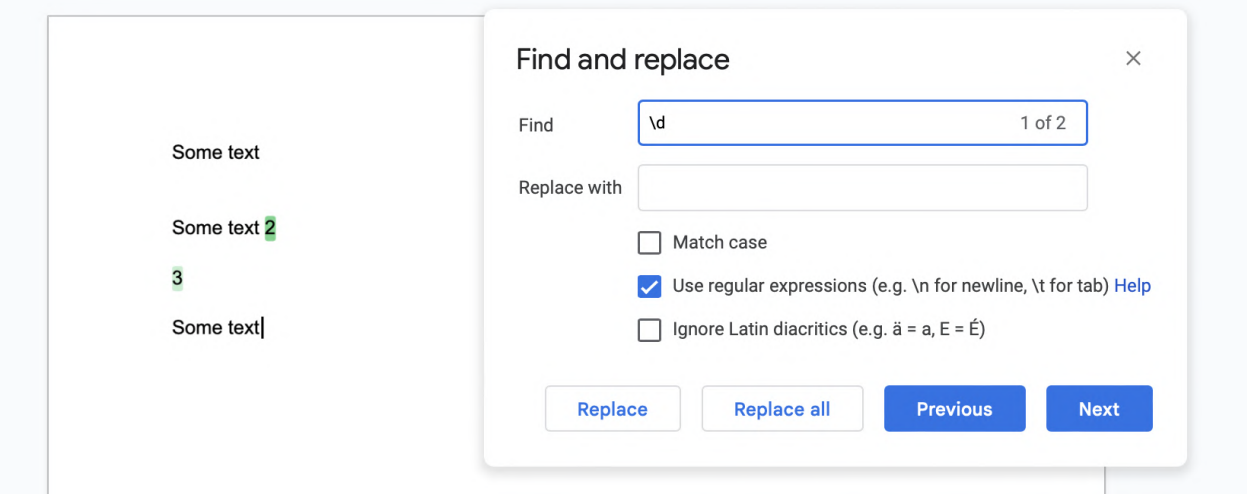
For scraping websites where the code changes a lot after the execution of JavaScript code in the browser, you can use such packages as Selenium (https://selenium-python.readthedocs.io). It allows you to use Python to open different browsers and simulate user actions in them.

Day 11. Regular expressions

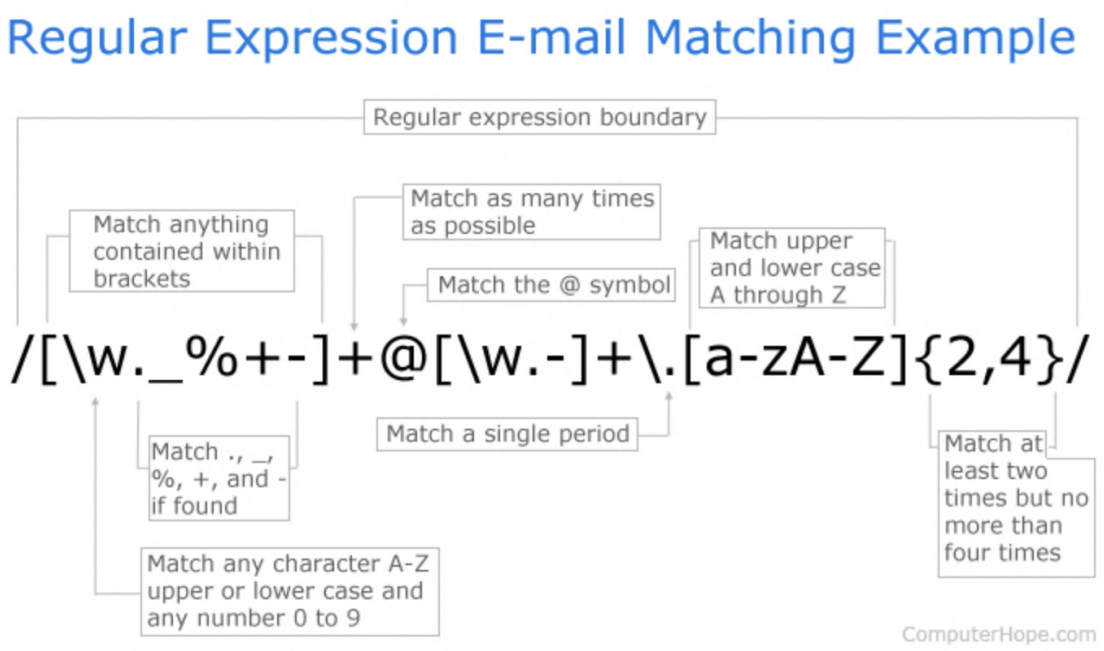
Regular expression is a sequence of characters that allows you to search for, retrieve and replace pieces of text in a source document that match certain patterns.

Regular expressions are supported by most popular programming languages and are used for searching, validating and extracting different data. For example, phone numbers, email addresses, IP addresses, cryptocurrency wallet numbers, etc.

One easy way to try regular expressions is to create a new Google Document, paste some text with letters and numbers in it, and then run Find and Replace.



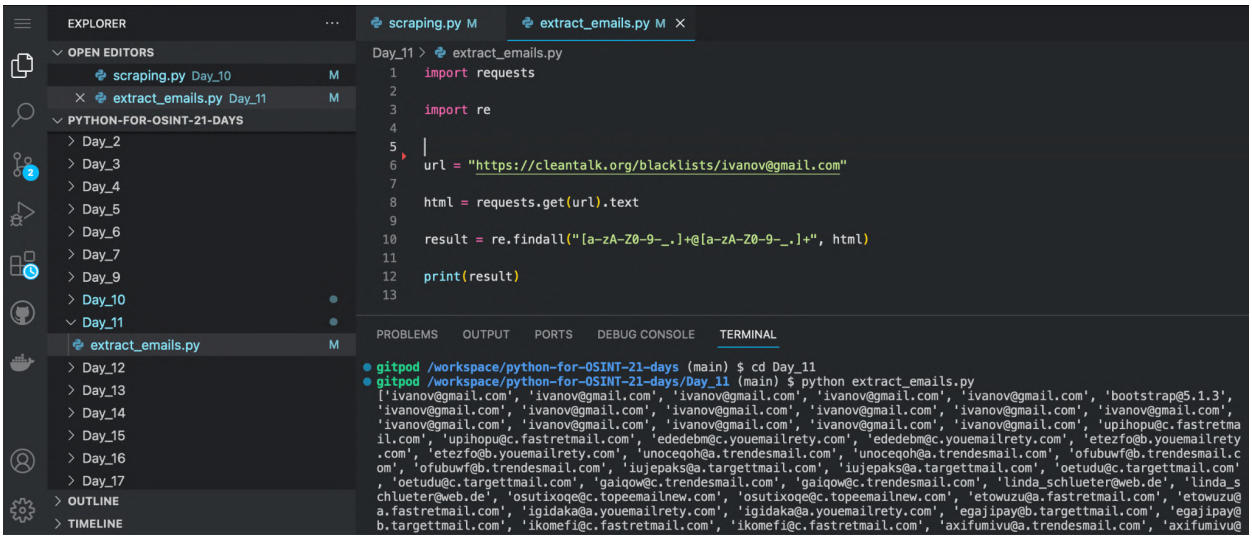
\d - allows you to find the digits. Here is a brief scheme that gives you a basic understanding of regular expression syntax:



Source of this image.

To work with regular expressions in Python we will use Re package (https://docs.python.org/3/library/re.html) It's available in Python by default.

Run extract_emails.py:



This will result in a list of all email addresses found on the https://cleantalk.org/blacklists/ivanov@gmail.com.

```
# Importing requests and re packages:

import requests
import re

# Create a variable with a link to the page we want to retrieve data from:

url = "https://cleantalk.org/blacklists/ivanov@gmail.com"

# Request the page and put the code into a variable:

html = requests.get(url).text

# Try to find email addresses in the code:

result = re.findall("[a-zA-Z0-9-_.]+@[a-zA-Z0-9-_.]+", html)

# Display the found email addresses:

print(result)
```

Try changing the URL variable and restarting the script.

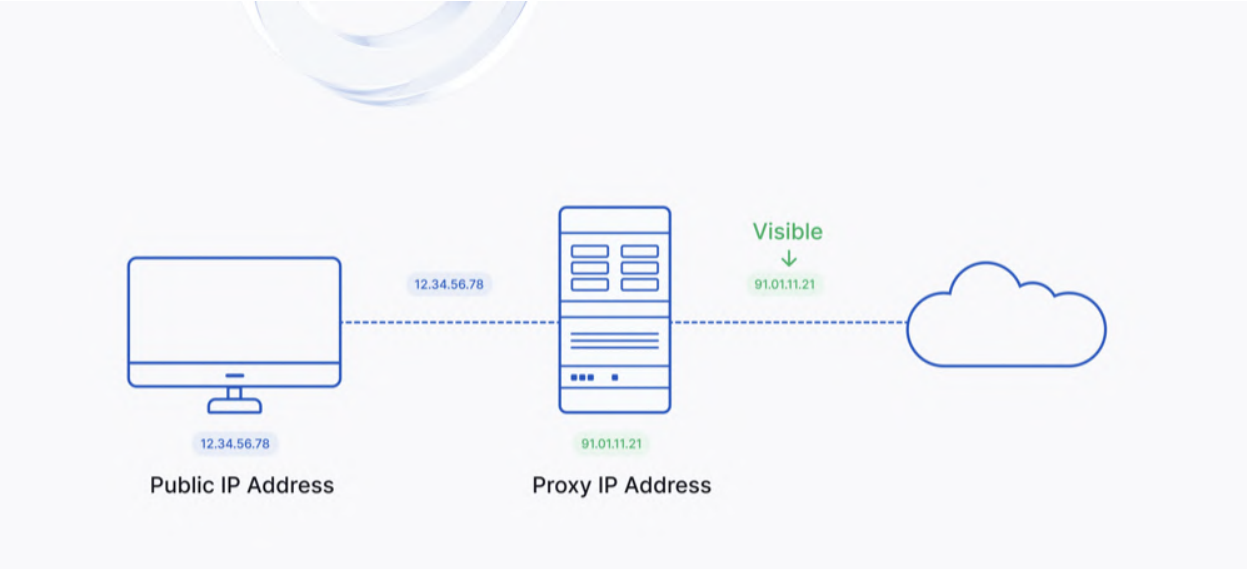
There is only one code example in this lesson, since your main goal today is to learn in detail how regular expressions are used in OSINT.

Please read this article on my Medium blog:

[How regular expressions can be useful in OSINT. Theory and some practice using Google Sheets]

Day 12. Proxies

Very many sites and services block IP addresses that send a large number of requests in a short time. You can bypass such protection by using Proxy servers (It doesn't always work, but sometimes it does).

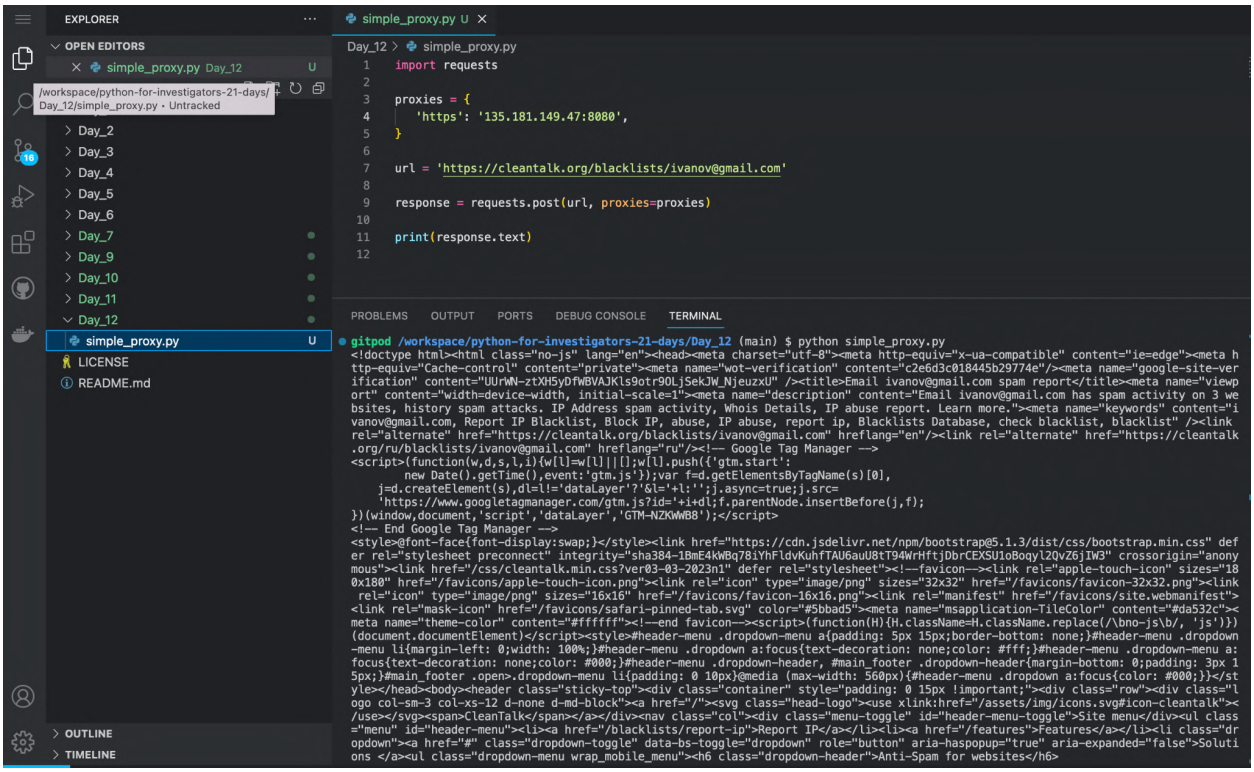


Thanks for picture, Upguard.com.

A **proxy server**, also known as an application-level gateway, can be a piece of software or a computer. Either way, it functions as a gateway between your device and the server you are connected to. It's like an ambassador that acts as your representative when transacting with different servers on the Web (Techslang).

Using them in Python is very easy. You just need to specify the address of the server you want to redirect traffic through when making a request.

Run simple_proxy.py:



Import requests package:

```
import requests
```

Create variable with https proxy server and port:

```
proxies = {
  'https': '135.181.149.47:8080',
}
```

Create variable with url for request:

```
url = 'https://cleantalk.org/blacklists/ivanov@gmail.com'
```

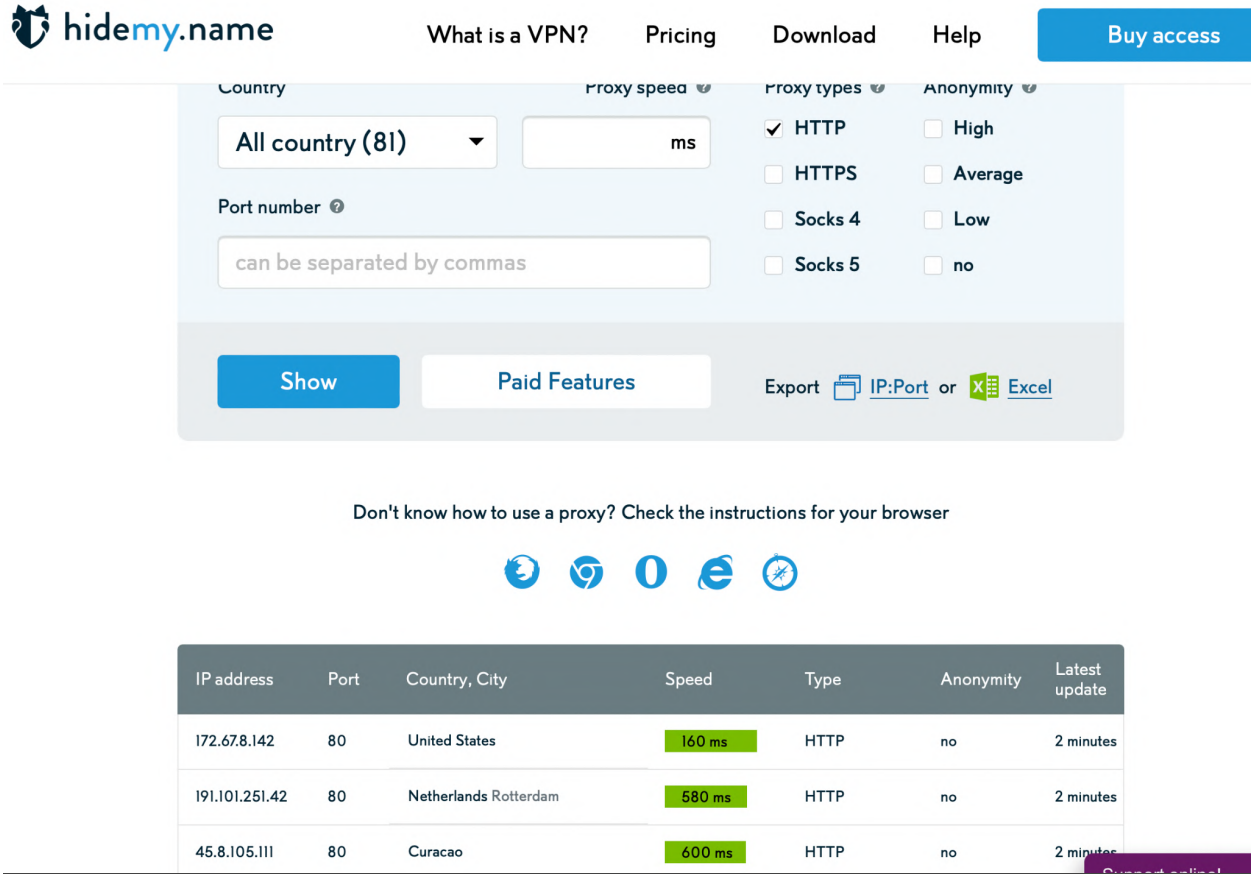
Make request through a proxy server:

```
response = requests.post(url, proxies=proxies)
```

Print text of web page:

```
print(response.text)
```

The proxy server used as an example in the code above is probably no longer working. So please replace it with another one. A huge number of free servers can be found on Google in a couple of seconds.



Examples of proxy servers list:

https://hidemy.name/en/

https://github.com/clarketm/proxy-list

https://github.com/TheSpeedX/PROXY-List

https://github.com/ShiftyTR/Proxy-List

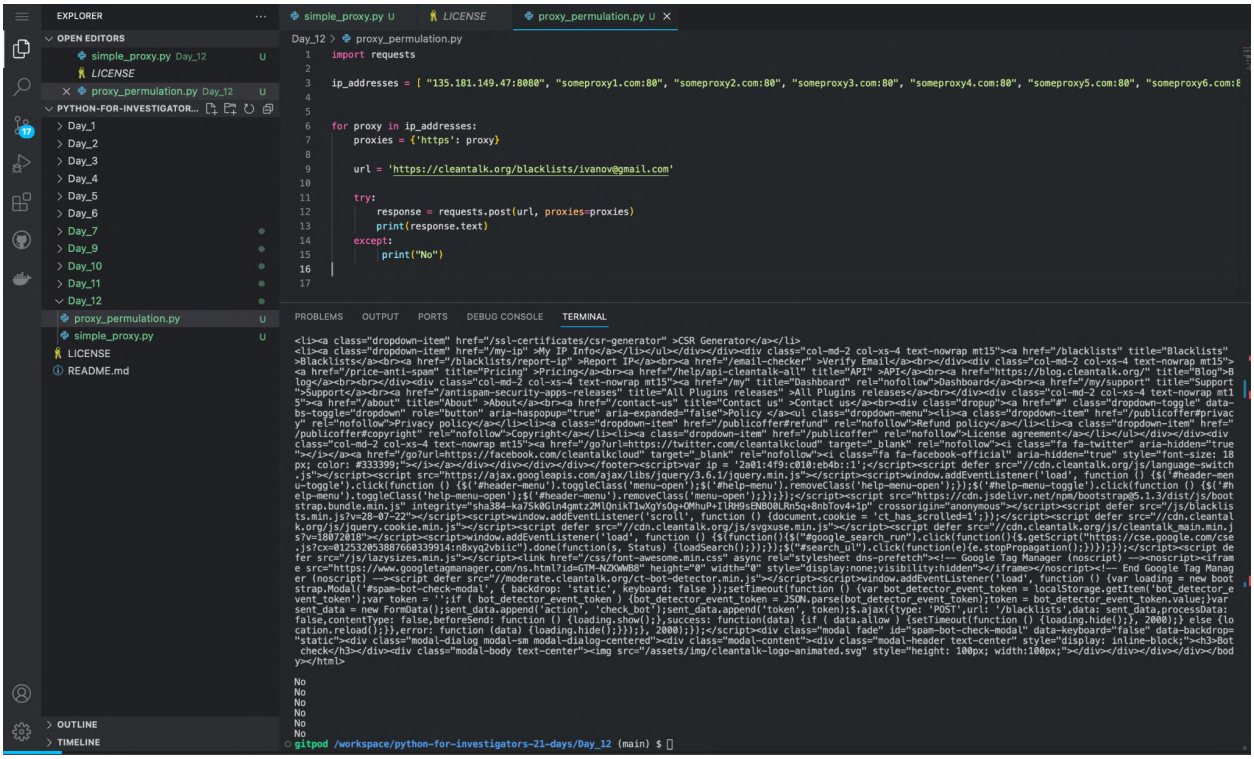
https://github.com/jetkai/proxy-list

One proxy server is not enough to successfully bypass scraping protections. After all, the target site can block them one by one and, in addition, free proxy servers can be extremely unstable.

Therefore, you may need to search through proxy servers in order to find one that works and is NOT blocked.

run proxy_permulation.py:

As in the first case, the proxy addresses on the list at the time of publication of the book may not work. Therefore, replace them with other ones (which, as said above, can be found in free lists) before start script.



Import requests package:

```
import requests
```

Create list with https proxy servers and ports:

```
ip_addresses = [ "135.181.149.47:8080", "someproxy1.com:80",
"someproxy2.com:80", "someproxy3.com:80", "someproxy4.com:80",
"someproxy5.com:80", "someproxy6.com:80"]
```

Create variable with url for request:

```
  url = 'https://cleantalk.org/blacklists/ivanov@gmail.com'
```

Go through ip_addresses list:

```
for proxy in ip_addresses:
  proxies = {'https': proxy}
```

Try to make request and print results:

```
  try:
      response = requests.post(url, proxies=proxies)
      print(response.text)
  except:
       print("No")
```

You can also use out-of-the-box tools to redirect traffic through proxy servers:

XX-net

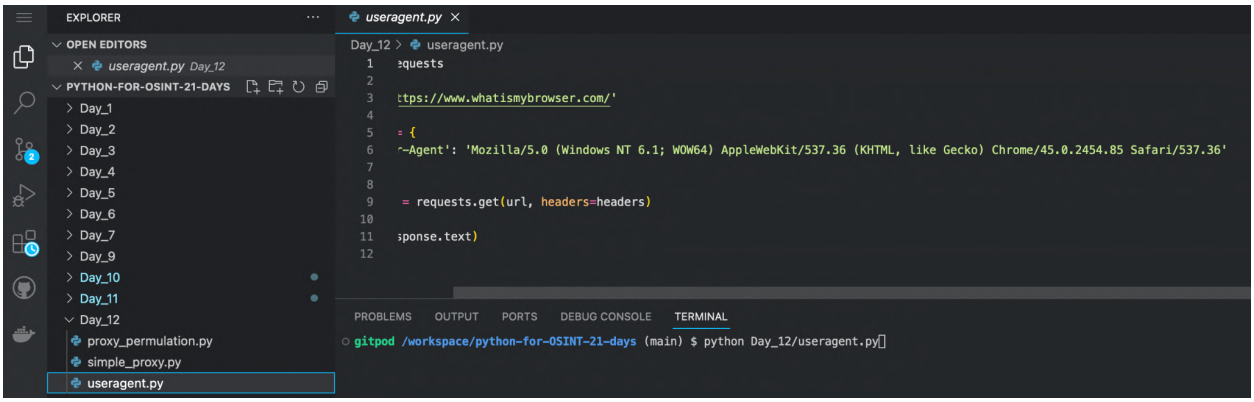
mitmproxy

Proxify (very good Go written tool from Projectdiscovery)

Often, simply changing your IP address is not enough to bypass blocked sites. You also have to add additional parameters to the request in the headers of the request. The settings will be individual for each task.

Here is a simple example of a request with the User-Agent header (information about the user's device) added.

Run useragent.py:



Import request package:

```
import requests
```

Create variable with link to website:

```
url = 'https://www.whatismybrowser.com/'
```

Create list with request headers (now we use only User-Agent header):

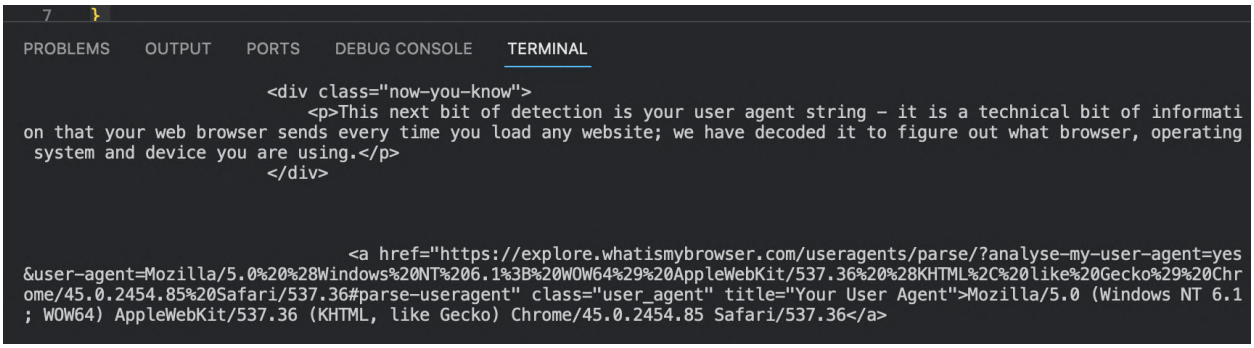
```
headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.85
Safari/537.36'
}
```

Send request:

```
response = requests.get(url, headers=headers)
```

Print response text:

```
print(response.text)
```



As a result, the html code of the page should be displayed, which will contain the User-Agent specified in the headers passed along with the request.

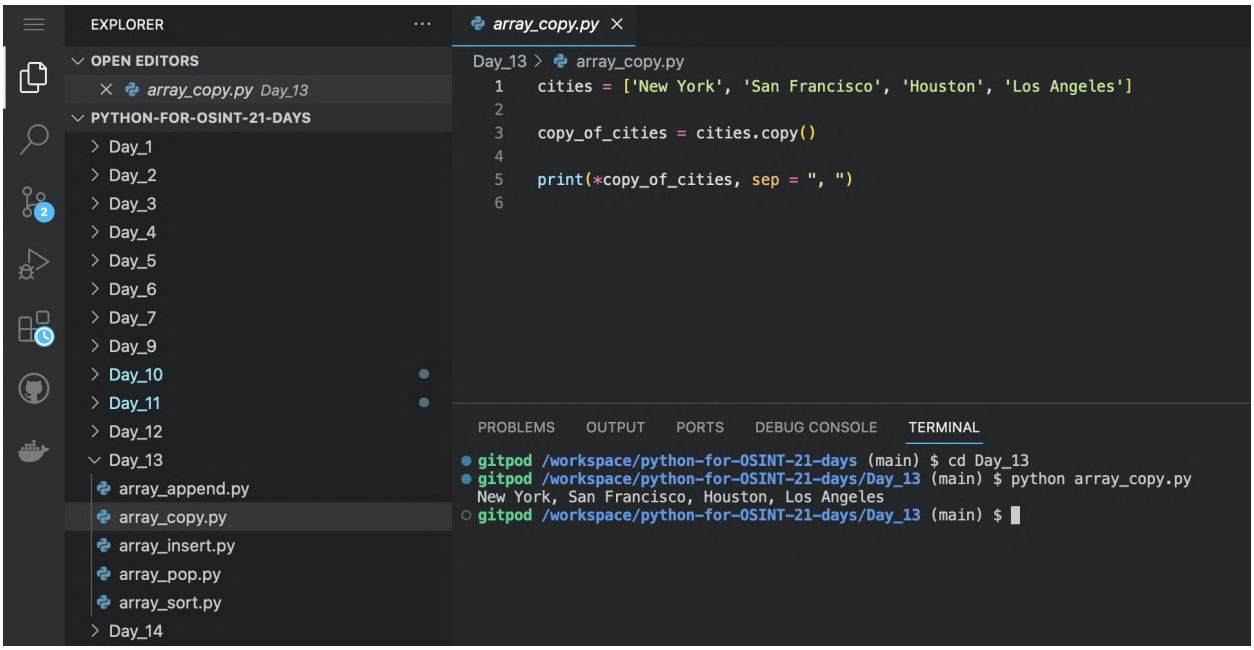
Day 13. Functions for working with lists

As you have already noticed, lists are a very important element of Python syntax that we have used in most of our lessons.

Today's lesson will be something of a rest day. We'll take a look at some very simple but very useful functions for working with lists.

First, let's learn how to copy arrays.

Run array_copy.py:



Create list of USA cities:

```
cities = ['New York', 'San Francisco', 'Houston', 'Los Angeles']
```

Copy list of USA cities:

```
copy_of_cities = cities.copy()
```

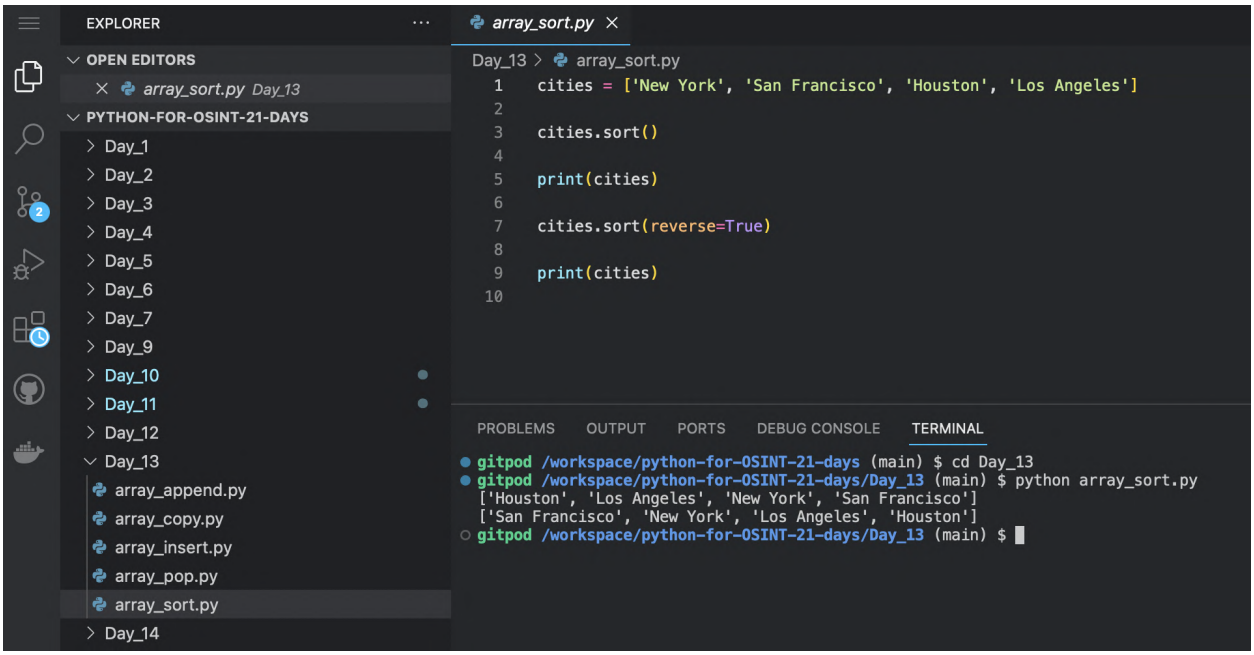
Print copy of list of USA cities with elements separated by “,”:

```
print(*copy_of_cities, sep = ", ")
```

Notice that we used a new way to output the array elements (we set up a separator).

Now, let's sort the list.

Run array_sort.py:



Create list of USA cities:

```
cities = ['New York', 'San Francisco', 'Houston', 'Los Angeles']
```

Sort list of USA cities by ascending:

```
cities.sort()
```

Print result:

```
print(cities)
```

Sort list of USA cities by descending :

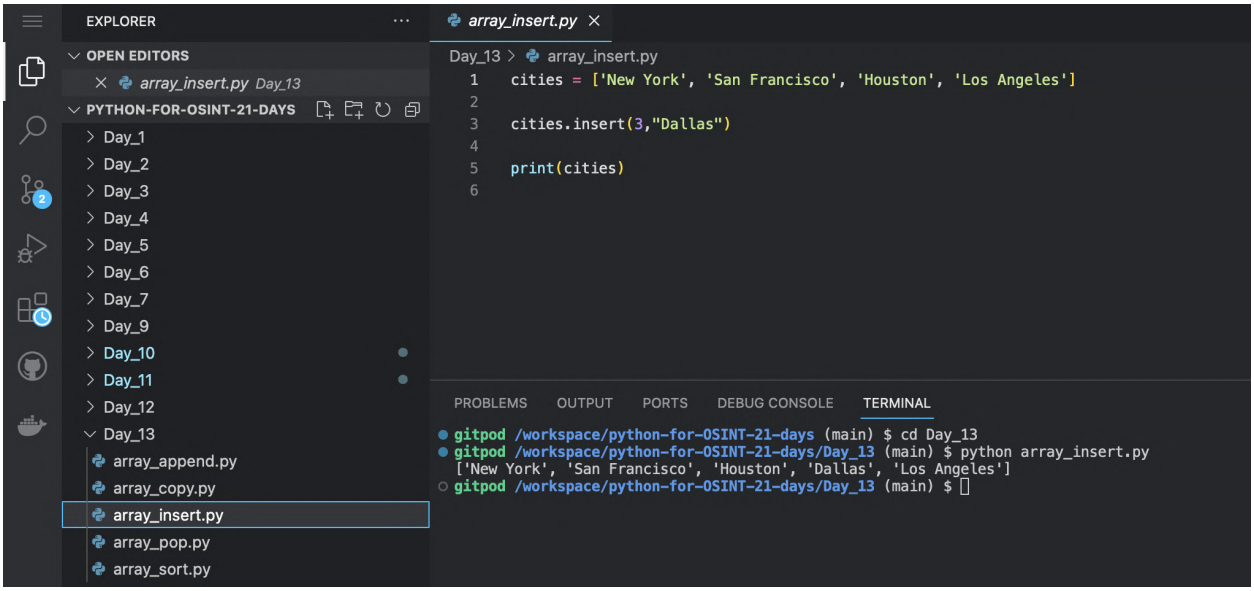
```
cities.sort(reverse=True)
```

Print results :

```
print(cities)
```

And here are two functions for adding elements to an array - insert() and append().

Run array_insert.py:



Create a list of U.S. city names:

```
cities = ['New York', 'San Francisco', 'Houston', 'Los Angeles']
```

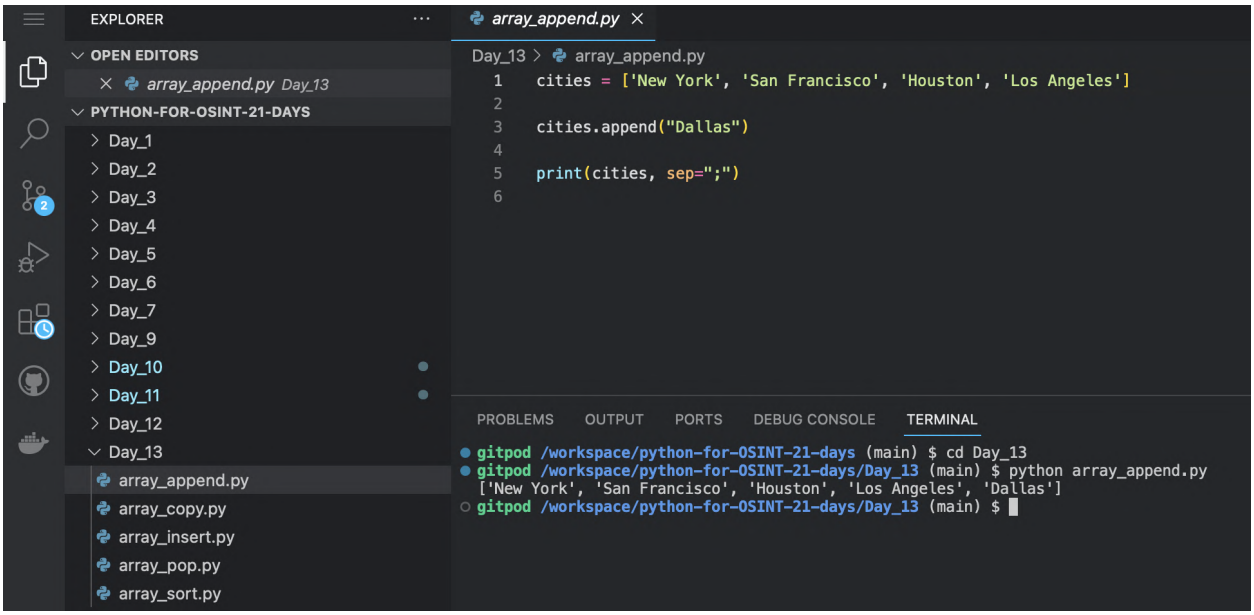
Add item number three and the text Dallas (remember that the count starts with zero):

```
cities.insert(3,"Dallas")
```

Display the modified list on the screen:

```
print(cities)
```

Run array_append.py:



Create a list of U.S. city names:

```
cities = ['New York', 'San Francisco', 'Houston', 'Los Angeles']
```

Add Dallas to the end of the list:

```
cities.append("Dallas")
```

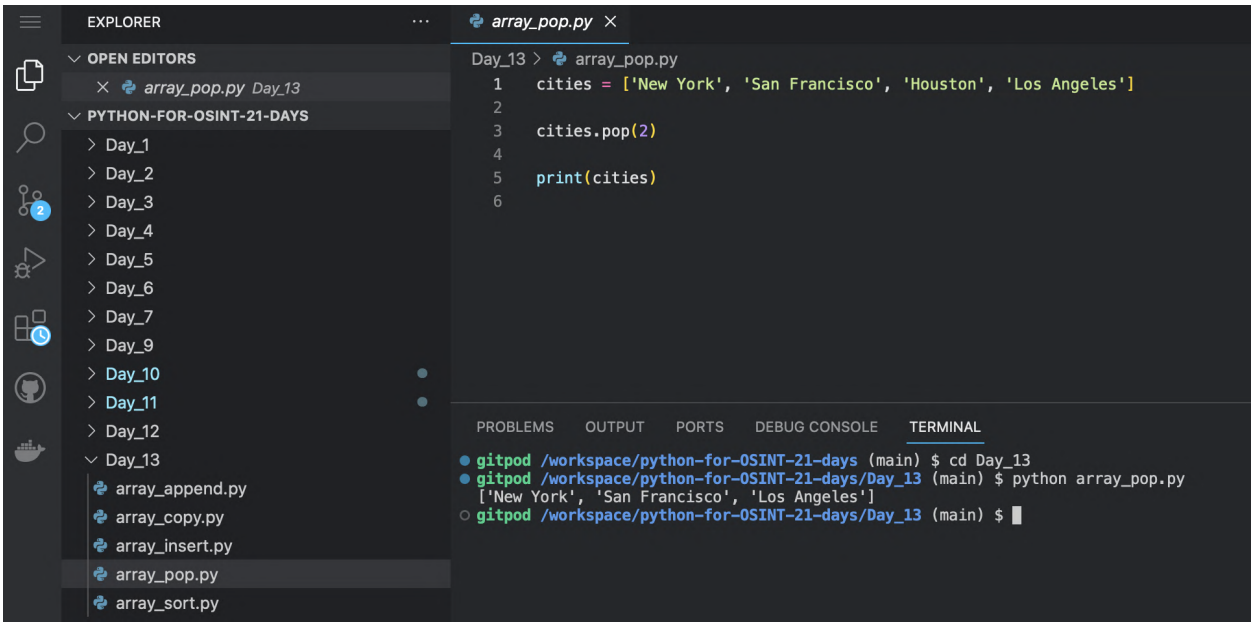
Display the list on the screen, separating elements with semicolons:

```
print(cities, sep=";")
```

Just in case, let me explain the difference between insert() and append(). Insert() inserts into an element at a certain location (under a certain number). And append() adds an element to the end of the array (under the last number).

Let's finish this lesson with a function that removes an element with a specific number from an array.

Run array_pop.py:



Create a list of U.S. city names:

```
cities = ['New York', 'San Francisco', 'Houston', 'Los Angeles']
```

Delete from it the element with the index two (remember that counting starts with zero):

```
cities.pop(2)
```

Display the changed list on the screen:

```
print(cities)
```

In this course we will return to the topic of arrays and discuss the most important function for working with them - map(), which is definitely worthy of its own lesson.

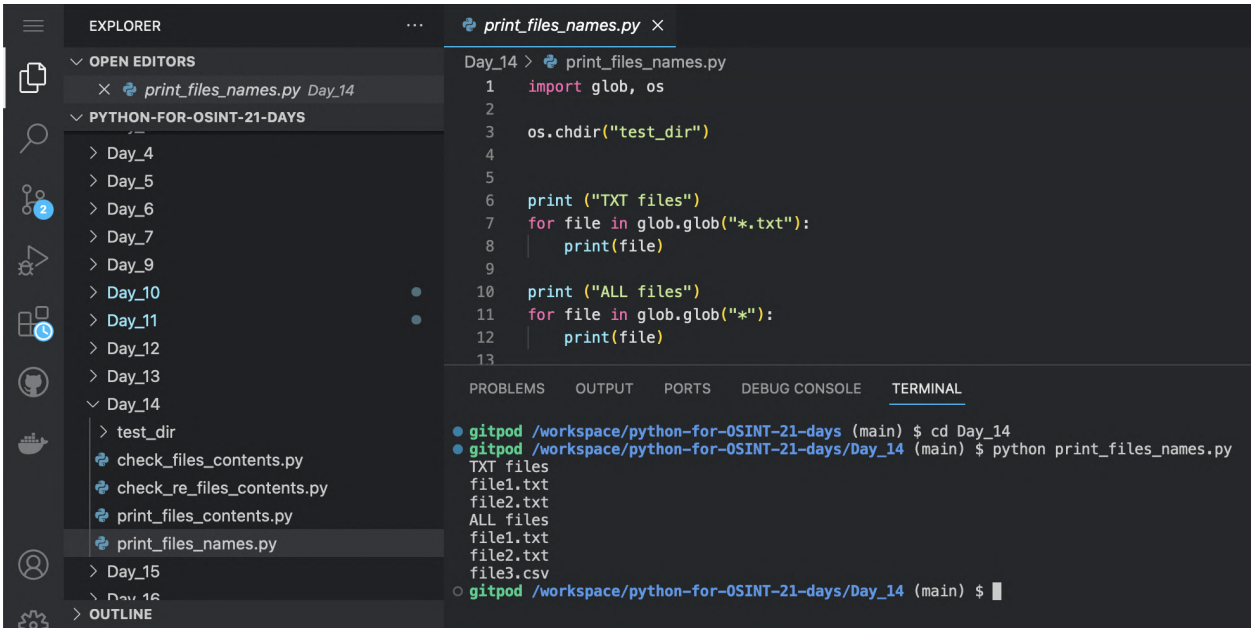
Day 14. Working with the file system

In this course we do not deal with databases and we use csv, json, txt files to store data. And sometimes you may want to write a script that will not process data from a single file with a specific name, but data from a large group of files (and sometimes located in different directories).

Therefore, you may need a minimal skills in working with the file system in Python.

We will use glob (https://docs.python.org/3/library/glob.html) and os (https://docs.python.org/3/library/os.html) packages to find and read files.

Run print_files_names.py:



Import glob and os packages:

```
import glob, os
```

Go to “test_dir” directory:

```
os.chdir("test_dir")
```

Print “TXT files” string:

```
print ("TXT files")
```

Search all files with txt extension and print it’s names:

```
for file in glob.glob("*.txt"):
    print(file)
```

Print “ALL files” string:

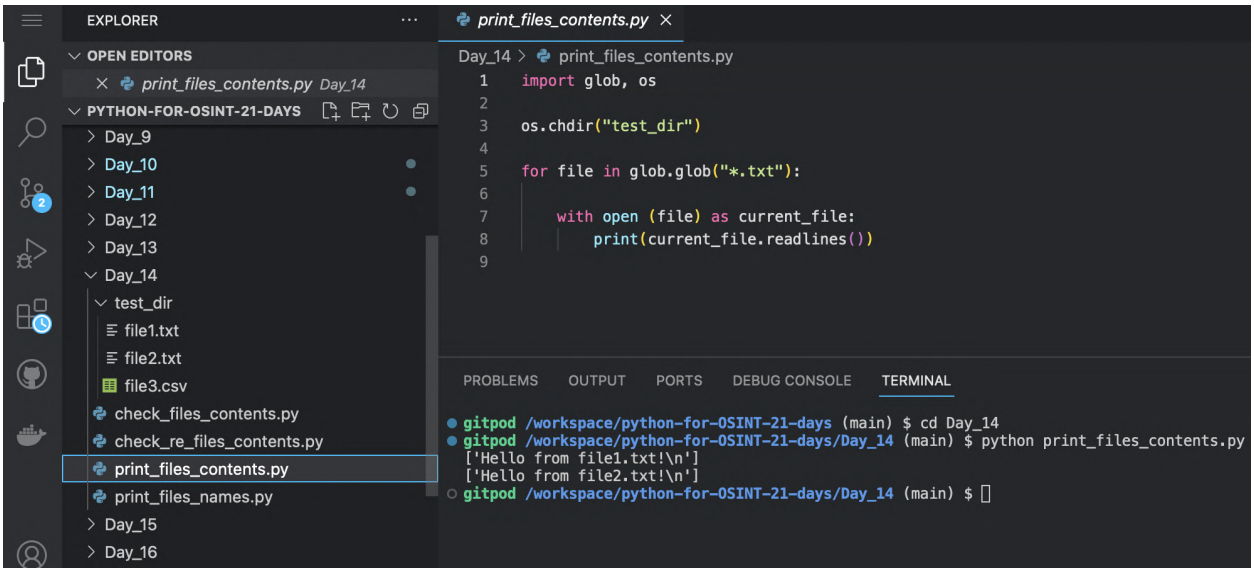
```
print ("ALL files")
```

Search all files and print it’s names:

```
for file in glob.glob("*"):
    print(file)
```

Now let's try to display the contents of the found files using the readlines() command.

Run print_files_contents.py:



Import glob and os packages:

```
import glob, os
```

Go to “test_dir” directory:

```
os.chdir("test_dir")
```

Search all files with txt extension and print it’s names:

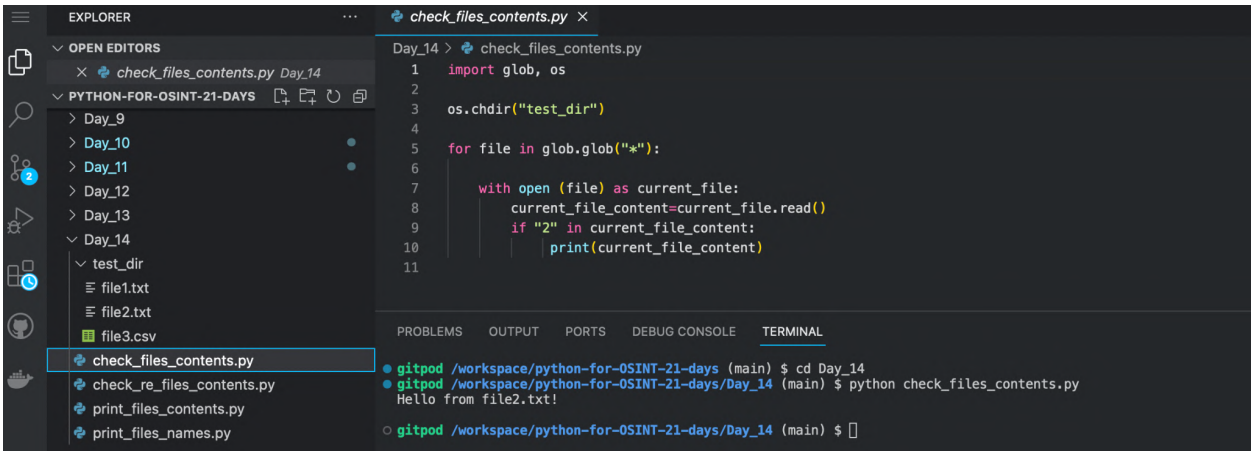
```
for file in glob.glob("*.txt"):
```

Open each files and print it’s content:

```
  with open (file) as current_file:
      print(current_file.readlines())
```

The contents of the files obtained in this way can be automatically analyzed. The simplest example is to check for the presence of any symbol or word.

Run check_files_contents.py:



Import glob and os packages:

```
import glob, os
```

Go to “test_dir” directory:

```
os.chdir("test_dir")
```

Go through all files

```
for file in glob.glob("*"):
```

Open each file:

```
  with open (file) as current_file:
```

Read content of each file:

```
      current_file_content=current_file.read()
```

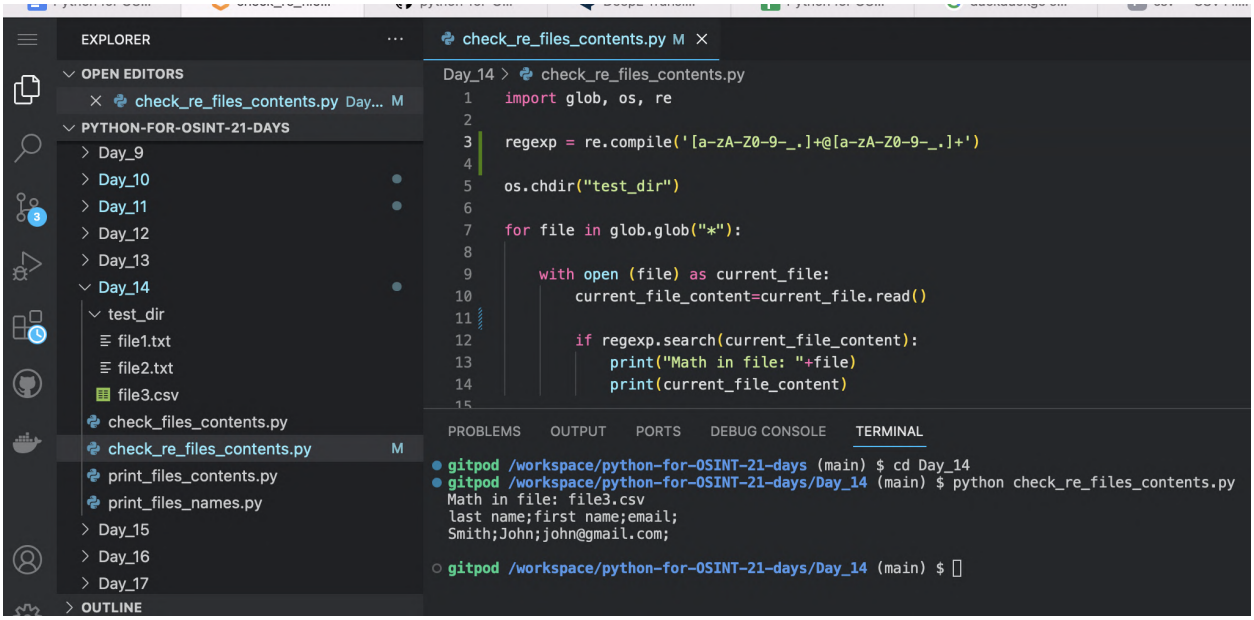
If file content include “2”, print it:

```
      if "2" in current_file_content:
```

```
            print(current_file_content)
```

Now, let's try to check each file for an email address in its text by using a regular expression (we learned them in lesson 11).

Run check_re_file_contents.py:



Import glob, os and re packages:

```
import glob, os, re
```

Create regex object:

```
regexp = re.compile('[a-zA-Z0-9-_.]+@[a-zA-Z0-9-_.]+')
```

Go to “test_dir” directory:

```
os.chdir("test_dir")
```

Go through all files

```
for file in glob.glob("*"):
```

Open each file:

```
  with open (file) as current_file:
```

Read content of file:

```
      current_file_content=current_file.read()
```

Check if the line corresponding to the regular expression can be found in the file text:

```
      if regexp.search(current_file_content):
```

If it is found, we display the file name and content:

```
          print("Math in file: "+file)
          print(current_file_content)
```

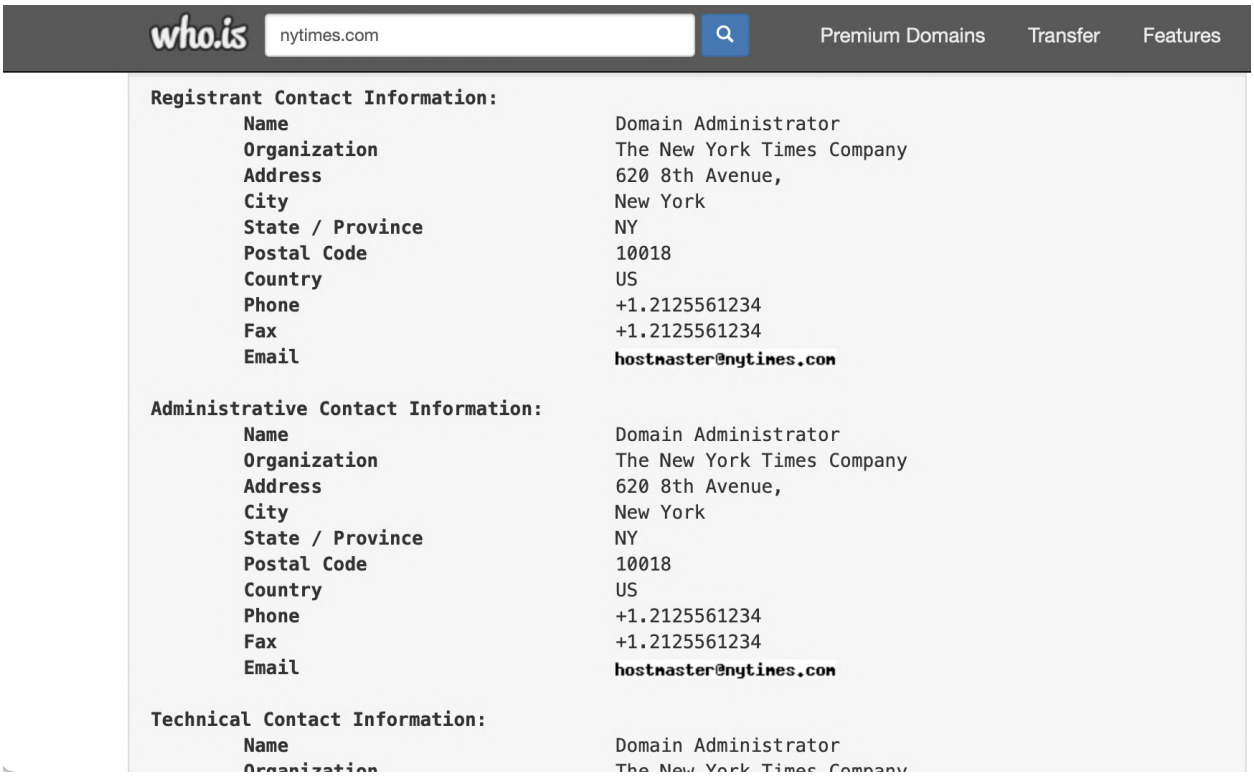
If you don't know where to get regular expressions for other types of data (phone numbers, IP addresses, etc.), it means that you weren't paying attention in Lesson 11 and didn't read the linked article.

Day 15. Domain information gathering

This lesson was the hardest to write because it is a topic worthy of a separate course called "Networking in Python" or "Python for Pentester.

Today I will try to explain the basic terms related to domain research and show you some Python packages that may be useful for this purpose.

WHOIS is a query and response protocol that provides information about existing domain names and all the pertinent data about them (Techslang)



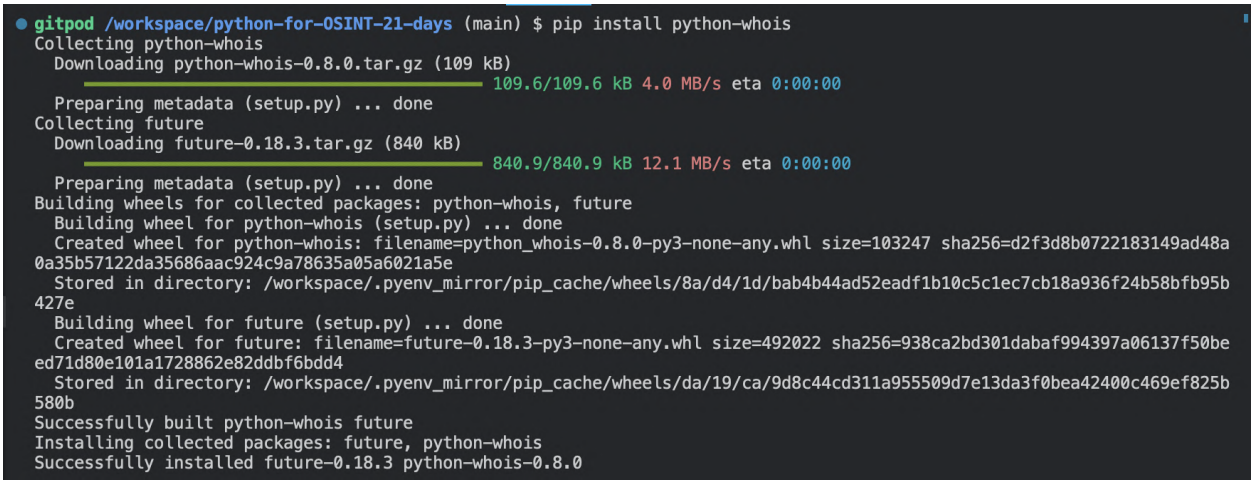
There are many free online services that display Whois data for a particular domain. For example:

https://who.is/whois/nytimes.com

There you can find out the date of registration of the domain, the date of expiration of the paid period of using the domain, the contact information of the company or person who currently owns the domain.

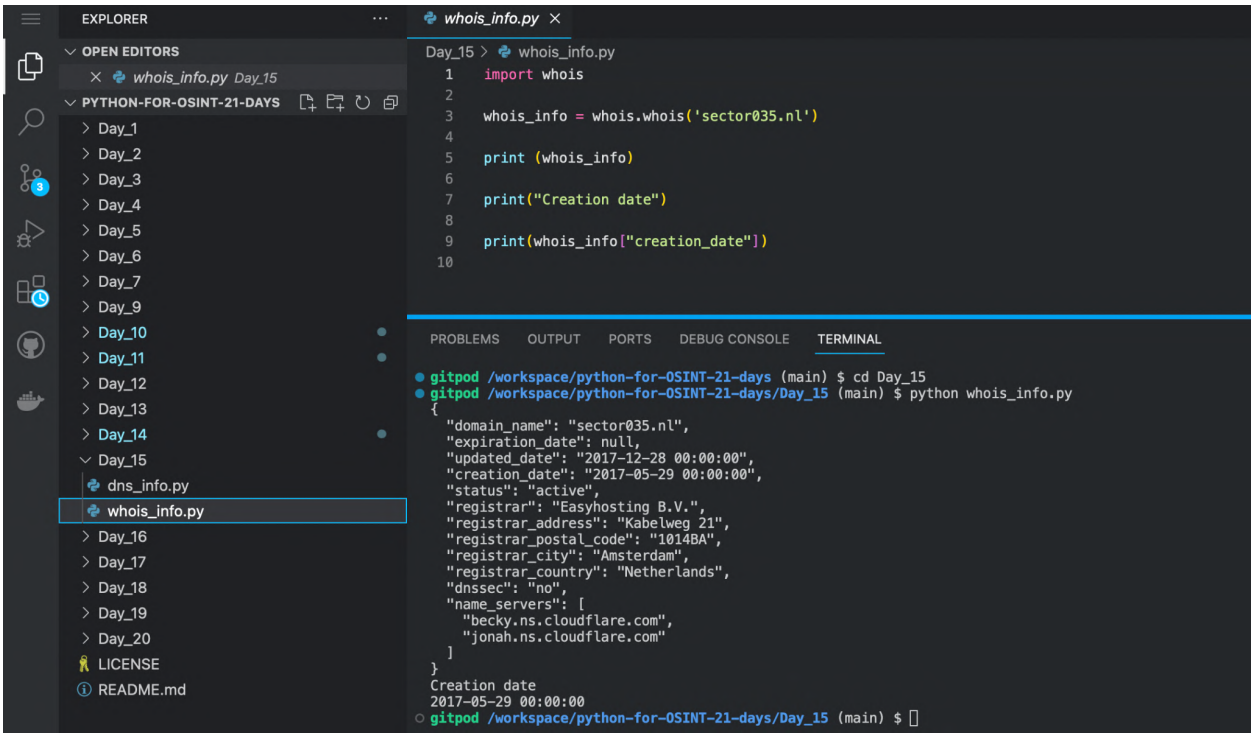
There are also services that allow you to find domains associated with a certain email (e.g. https://www.whoxy.com/) and see the history of whois data of the domain (https://research.domaintools.com/research/whois-history/).

Many packages have been created for Python to automate the handling of WHOIS data. Let's try Python-Whois package (https://pypi.org/project/python-whois/).



pip install python-whois

Run whois_info.py:



Import python-whois package:

```
import whois
```

Create variable with a target domain name:

```
whois_info = whois.whois('sector035.nl')
```

Print all whois info of this domain:

```
print (whois_info)
```

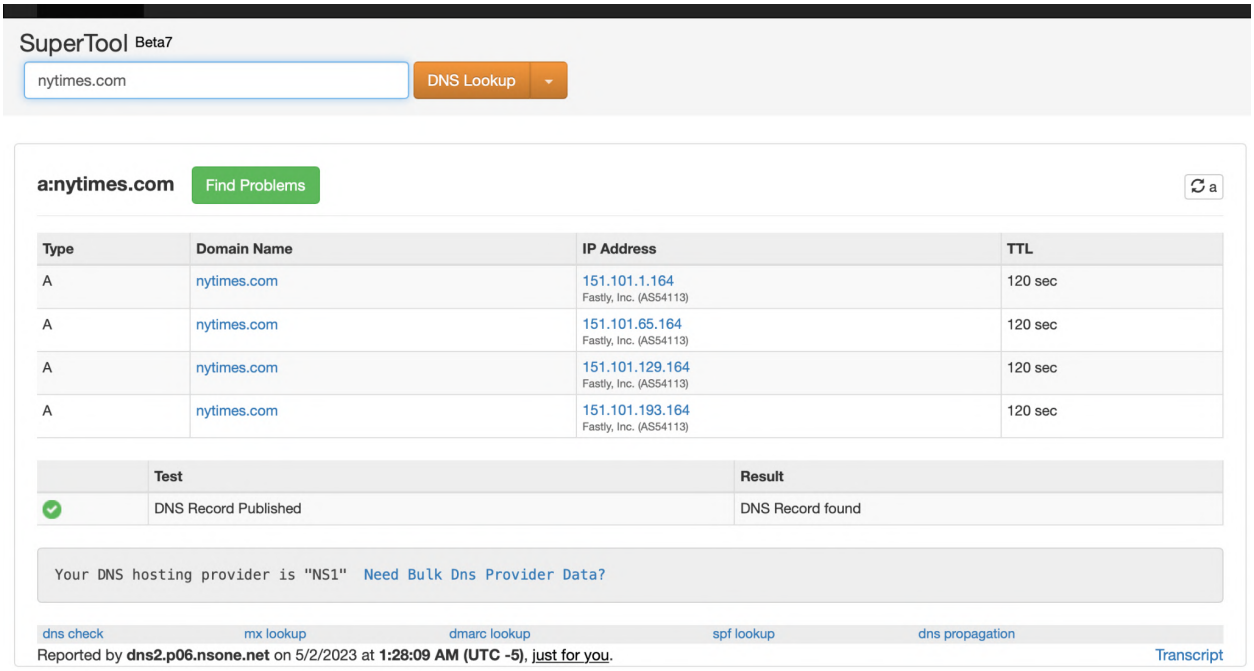
Print string “Creation date”::

```
print("Creation date")
```

Print creation date of target domain:

```
print(whois_info["creation_date"])
```

DNS (Domain Name System) is a system for naming computers on the Internet. It's a database that maps each numerical Internet address (called an IP address) with the corresponding domain name. (Techslang)



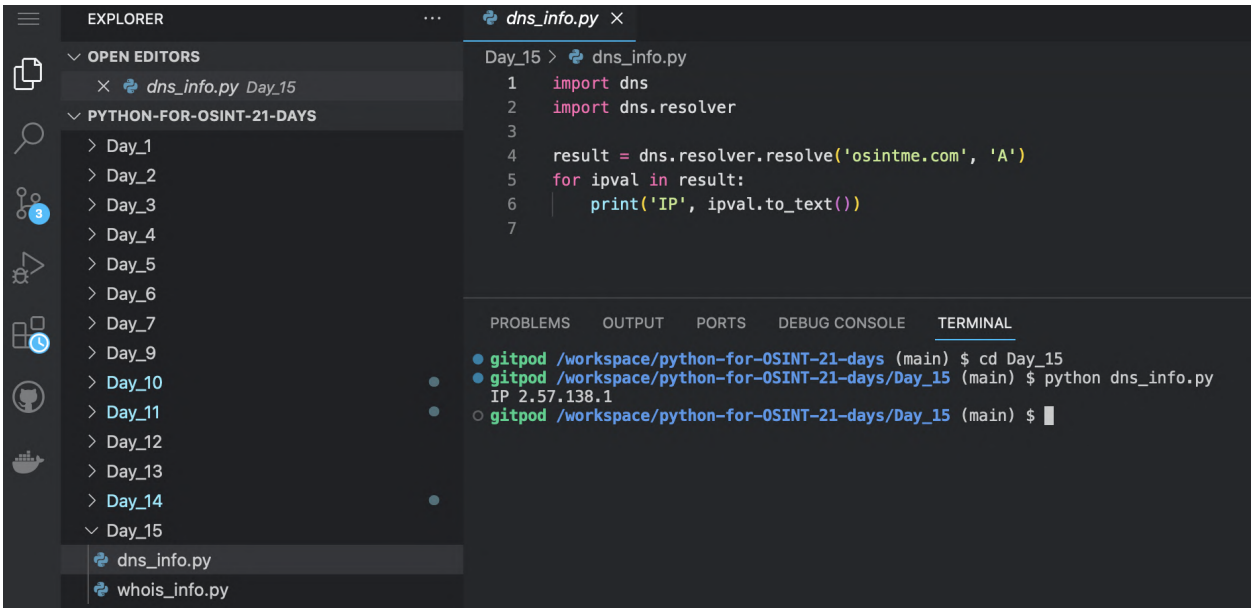
To see what the DNS data of the domain looks like you can use one of the free online services (example, https://mxtoolbox.com/).

You can use the DNSPython package to automate the retrieval of DNS data (https://pypi.org/project/dnspython/).

```
● gitpod /workspace/python-for-OSINT-21-days/Day_15 (main) $ pip install dnspython
  Collecting dnspython
    Downloading dnspython-2.3.0-py3-none-any.whl (283 kB)
       ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 283.7/283.7 kB 6.7 MB/s eta 0:00:00
  Installing collected packages: dnspython
  Successfully installed dnspython-2.3.0
```

pip install dnspython

Run python dns_info.py:



Import pythondns package and dns.resolver:

```
import dns
```

```
import dns.resolver
```

Get A records for osintme.com domains:

```
result = dns.resolver.resolve('osintme.com', 'A')
```

Go through results and print each of them:

```
for ipval in result:
```

```
    print('IP', ipval.to_text())
```

When collecting data about a site, it can be useful to find its subdomains in order to use them as an additional source of information.

Let's see how the Discosub (https://pypi.org/project/discosub/) tool works.

```
● gitpod /workspace/python-for-OSINT-21-days (main) $ pip install discosub
  Collecting discosub
    Downloading discosub-0.3.0-py2.py3-none-any.whl (399 kB)
       ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 399.5/399.5 kB 6.6 MB/s eta 0:00:00
  Collecting click
    Downloading click-8.1.3-py3-none-any.whl (96 kB)
       ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 96.6/96.6 kB 8.9 MB/s eta 0:00:00
  Installing collected packages: click, discosub
  Successfully installed click-8.1.3 discosub-0.3.0
```

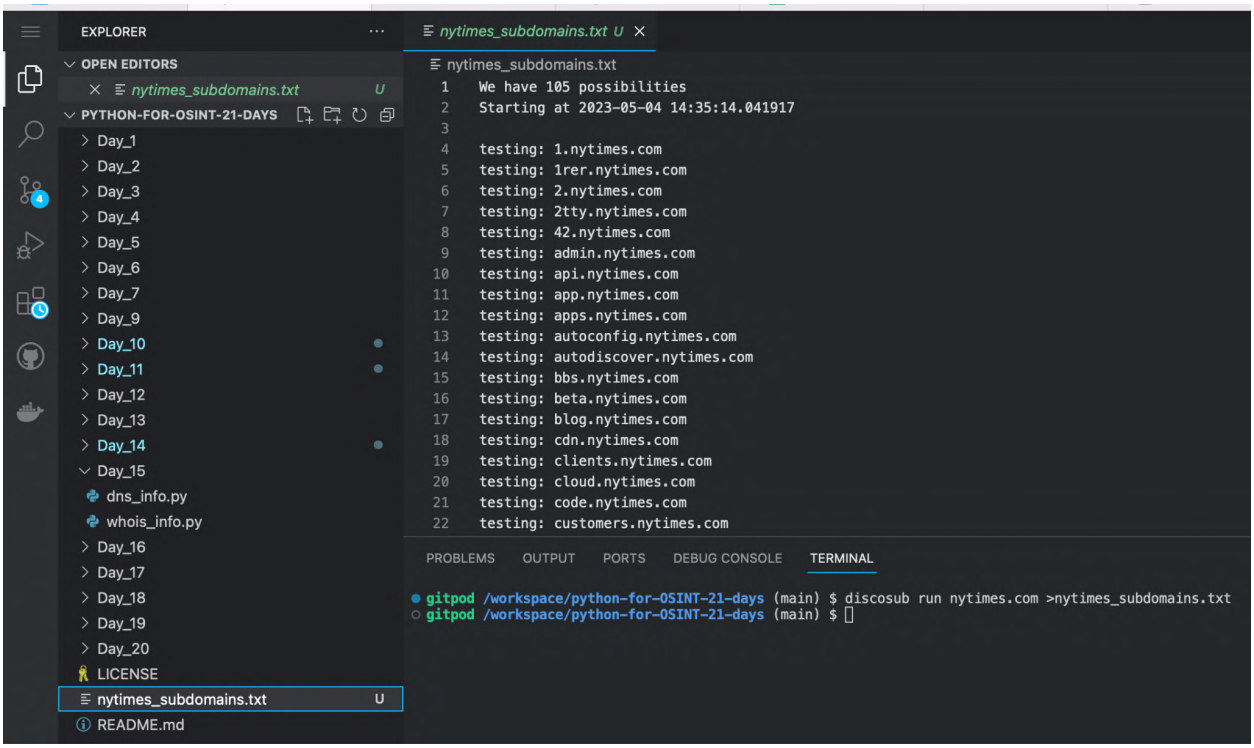
Install Discosub package from pip:

```
pip install discosub
```

```
● gitpod /workspace/python-for-OSINT-21-days (main) $ discosub run nytimes.com
  We have 105 possibilities
  Starting at 2023-05-04 14:34:18.667609

  testing: 1.nytimes.com
  testing: 1rer.nytimes.com
  testing: 2tty.nytimes.com
  testing: admin.nytimes.com
  testing: app.nytimes.com
  testing: autoconfig.nytimes.com
  testing: bbs.nytimes.com
  testing: blog.nytimes.com
  testing: clients.nytimes.com
  testing: code.nytimes.com
  testing: cvs.nytimes.com
  testing: doc.nytimes.com
  testing: email.nytimes.com
  testing: f.nytimes.com
  testing: feed.nytimes.com
  testing: feeds.nytimes.com
  testing: 2.nytimes.com
  testing: api.nytimes.com
  testing: autodiscover.nytimes.com
  testing: beta.nytimes.com
  testing: cloud.nytimes.com
  testing: dev.nytimes.com
  testing: files.nytimes.com
  testing: 42.nytimes.com
  testing: cdn.nytimes.com
  testing: customers.nytimes.com
  testing: exchange.nytimes.com
  testing: finance.nytimes.com
  testing: events.nytimes.com
  testing: apps.nytimes.com
  testing: forum.nytimes.com
  testing: forums.nytimes.com
  testing: ftp.nytimes.com
  testing: git.nytimes.com
```

Run Discosub for nytimes.com domain:

```
discosub run nytimes.com
```



There is also an option to run Discosub with the results saved to the nytimes_subdomains.txt file:

```
discosub run nytimes.com >nytimes_subdomains.txt
```

It is worth clarifying that Discosub is good in its simplicity and is therefore chosen as an example. But in terms of the quality of the results obtained it lags behind similar tools (finds fewer subdomains).

If your task is to find the maximum number of subdomains for a particular domain (or preferably all of them), then I recommend that you use several tools simultaneously. For example:

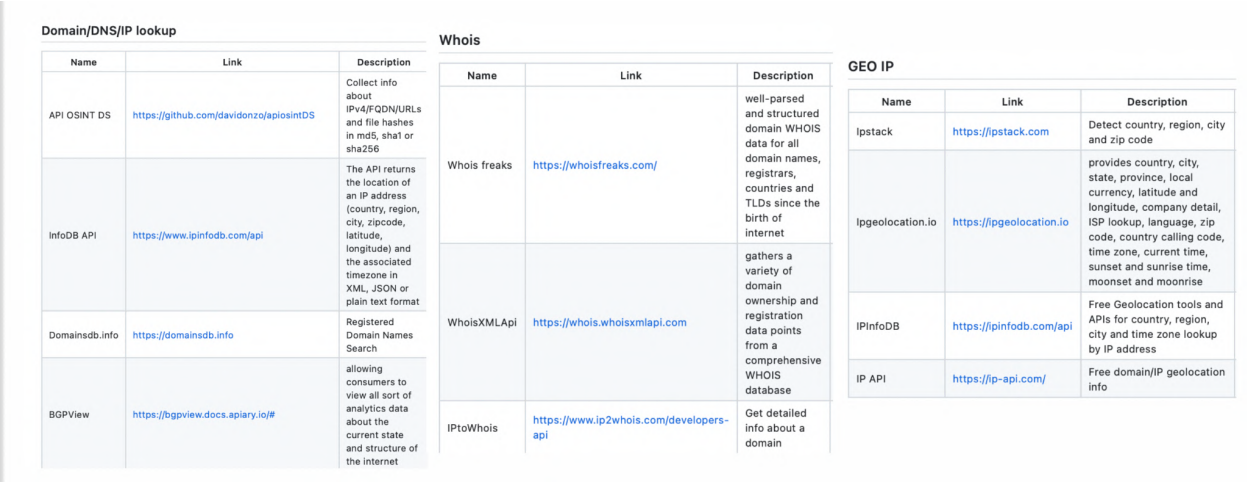
Sublist3r

SubDomainizer

Subscraper

Subfinder (written in Go language very good tool from Projectdiscovery)

and many others.



Domain/DNS/IP lookup

Name	Link	Description
API OSINT DS	https://github.com/davidonzo/apiosintDS	Collect info about IPv4/FQDN/URLs and file hashes in md5, sha1 or sha256
InfoDB API	https://www.ipinfodb.com/api	The API returns the location of an IP address (country, region, city, zipcode, latitude, longitude) and the associated timezone in XML, JSON or plain text format
Domainsdb.info	https://domainsdb.info	Registered Domain Names Search
BGPView	https://bgpview.docs.apiary.io/#	allowing consumers to view all sort of analytics data about the current state and structure of the internet

Whois

Name	Link	Description
Whois freaks	https://whoisfreaks.com/	well-parsed and structured domain WHOIS data for all domain names, registrars, countries and TLDs since the birth of internet
WhoisXMLApi	https://whois.whoisxmlapi.com	gathers a variety of domain ownership and registration data points from a comprehensive WHOIS database
IPtoWhois	https://www.ip2whois.com/developers-api	Get detailed info about a domain

GEO IP

Name	Link	Description
Ipstack	https://ipstack.com	Detect country, region, city and zip code
Ipgeolocation.io	https://ipgeolocation.io	provides country, city, state, province, local currency, latitude and longitude, company detail, ISP lookup, language, zip code, country calling code, time zone, current time, sunset and sunrise time, moonset and moonrise
IPInfoDB	https://ipinfodb.com/api	Free Geolocation tools and APIs for country, region, city and time zone lookup by IP address
IP API	https://ip-api.com/	Free domain/IP geolocation info

Also do not forget that there are many APIs for domain information gathering, on the basis of which you can create your own scripts, ideally suited for the purposes of your investigation (we talked about it in more detail in lessons 5 and 6).

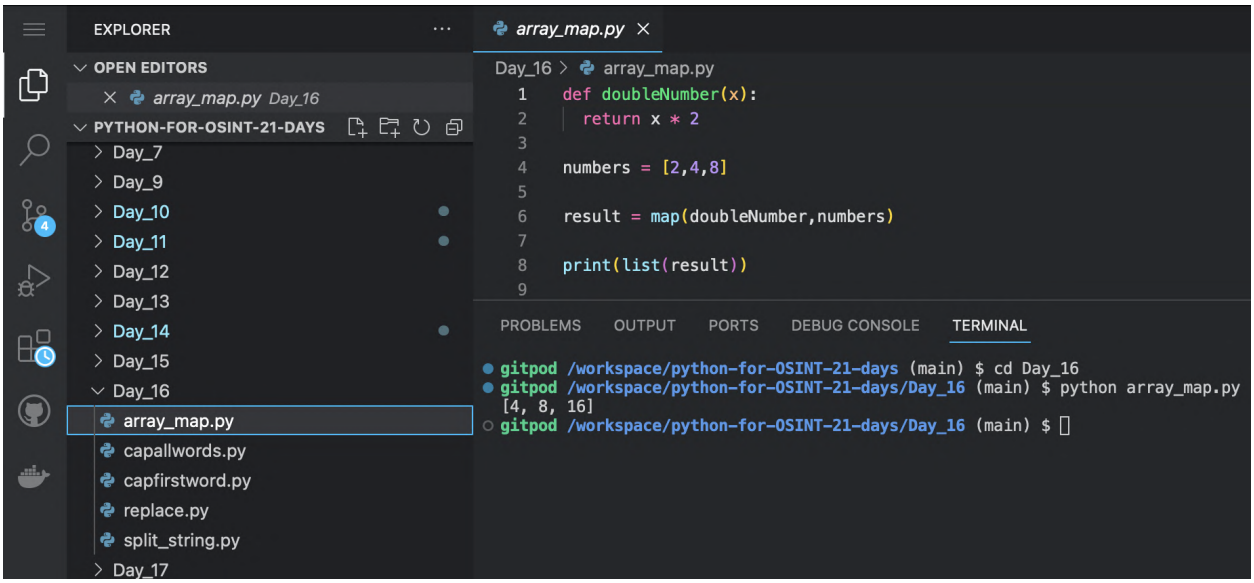
API-s-for-OSINT (Domain/DNS/IP lookup)

Day 16. List mapping and functions for work with strings

map() is a function that allows to apply some other function to each element of a particular list. It opens up huge opportunities for us to work with data.

For example, we can make some calculations with each element of the list, replace or add some characters in a group of strings, convert some values to other values, decrypt hashes. What to say... You can use map() with almost any other function.

Let's see how it works. Run array_map.py:



Create a function that multiplies the number passed to it by two:

```
def doubleNumber(x):
 return x * 2
```

Create list of three numbers:

```
numbers = [2,4,8]
```

Apply the doubleNumbers function one by one to each element of the list of numbers:

```
result = map(doubleNumber,numbers)
```

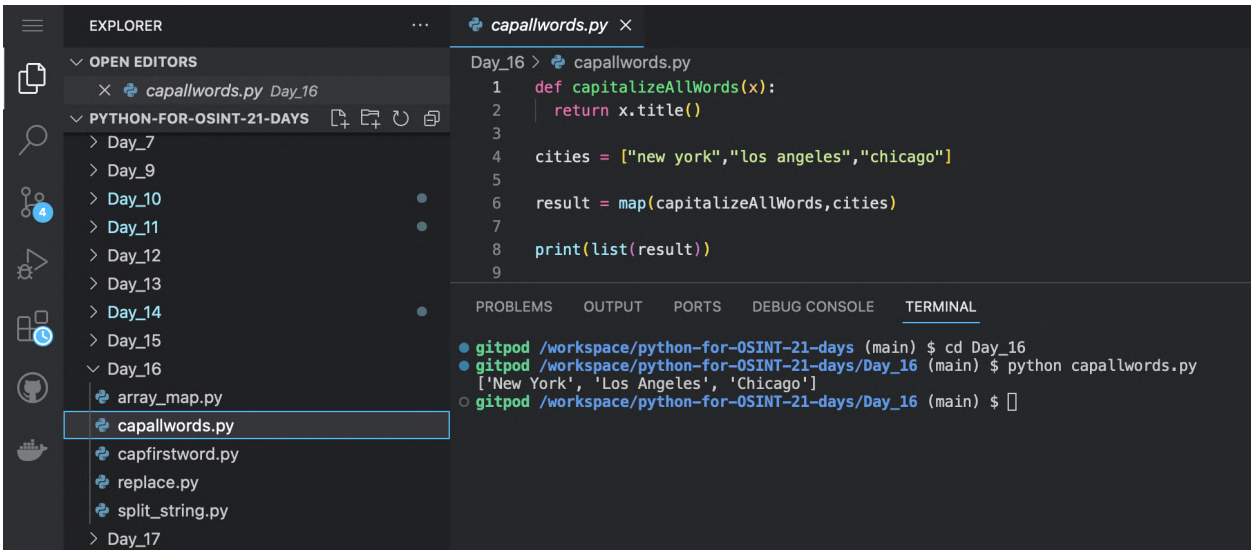
Display the modified list:

```
print(list(result))
```

As I consider the map() function to be very important, there will be many examples of its use in this lesson. I'll also show you some Python functions for working with strings.

Let's try making the first letters of the words capitalised in each element of the list.

Run capallwords.py:



Create a function that capitalizes the first letters in all words that are passed to it:

```
def capitalizeAllWords(x):
 return x.title()
```

#Create a list of USA cities:

```
cities = ["new york","los angeles","chicago"]
```

Apply the capitalizeAllWords function one by one to each element of the list of USA cities:

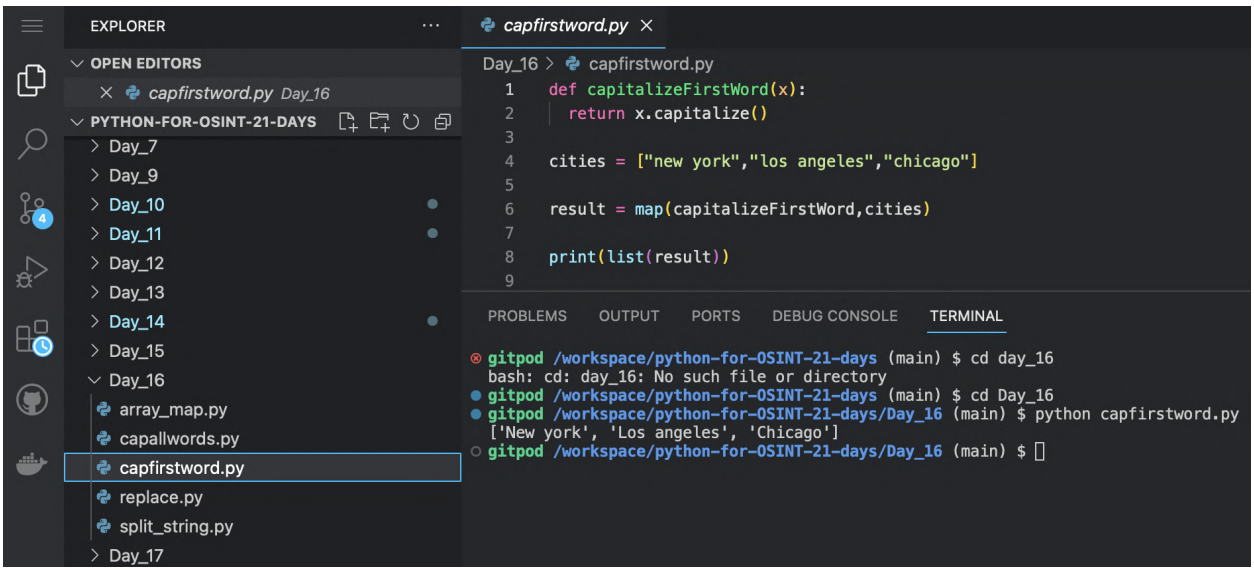
```
result = map(capitalizeAllWords,cities)
```

Display the modified list:

```
print(list(result))
```

Now let's capitalize only the first letter of each element.

Run capfirstword.py:



Create a function capitalizerDirstWord that changes the first letter in a line into a capital letter:

```
def capitalizeFirstWord(x):
 return x.capitalize()
```

Create a list of US cities:

```
cities = ["new york","los angeles","chicago"]
```

Run map() function for the list of three U.S. cities and capitalizeFirstWord function:

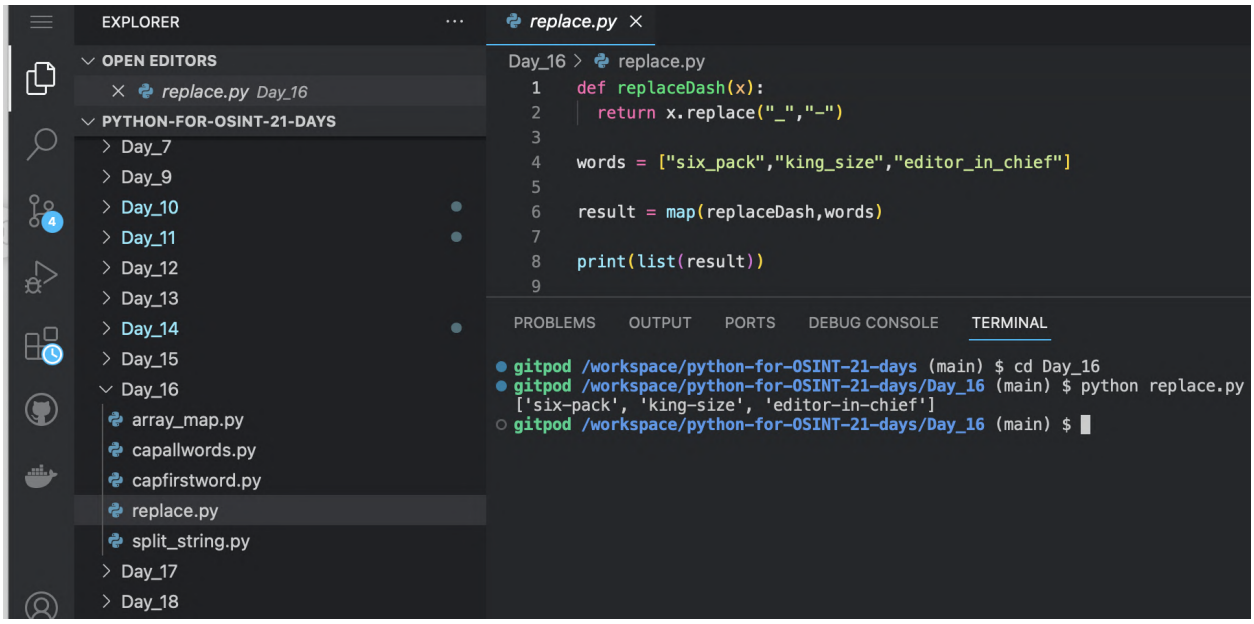
```
result = map(capitalizeFirstWord,cities)
```

Print the result on the screen:

```
print(list(result))
```

Another very simple but very useful skill is to replace some characters in a string with others.

Run replace.py:



Create a replaceDash function that replaces the underscore with the normal underscore:

```
def replaceDash(x):
    return x.replace("_","-")
```

Create a list of three words:

```
words = ["six_pack","king_size","editor_in_chief"]
```

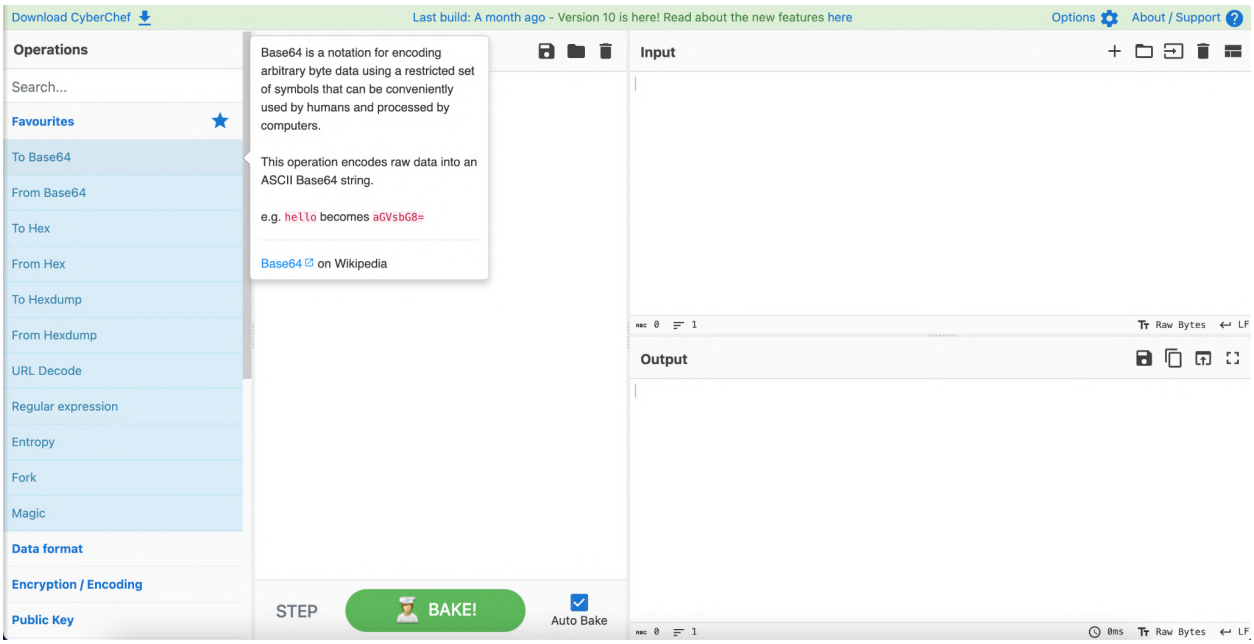
Run the map() function with arguments in the form of an array of three words and the replaceDash function:

```
result = map(replaceDash,words)
```

Print the results on the screen:

```
print(list(result))
```

Python has many other functions for working with strings and you can also install additional packages to extend this functionality.



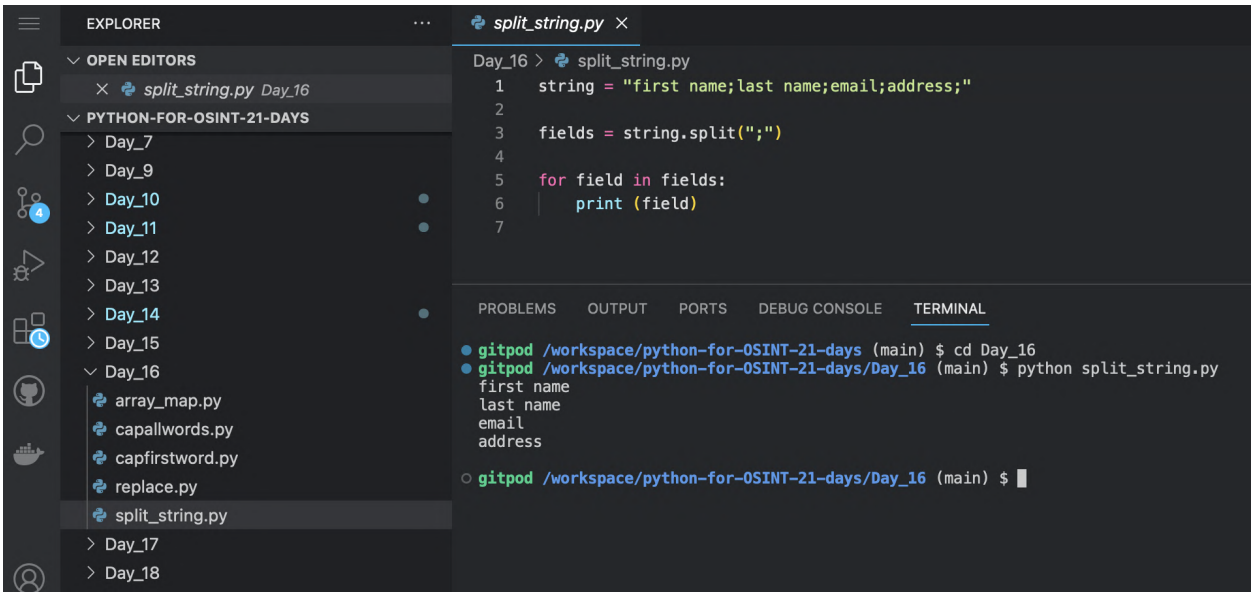
For example, the Python version of the well-known online data converter Cyber Shef (https://pypi.org/project/chepy/). It can:

decode URLs;
decode/encode base64;
convert Binary to Decimal and vice versa;
PGP encrypt/decrypt.

and more.

And at the end of this lesson, I want to show another simple but very important function that knows how to split one line into several according to the delimiter character.

Run split_string.py:



Create a line of multiple lines, separated by semicolons:

```
string = "first name;last name;email;address;"
```

Start the split() function, using the semicolon as its argument:

```
fields = string.split(";")
```

Go through the elements of the array in a loop and display each one in turn:

```
for field in fields:
  print (field)
```

Day 17. Generating documents

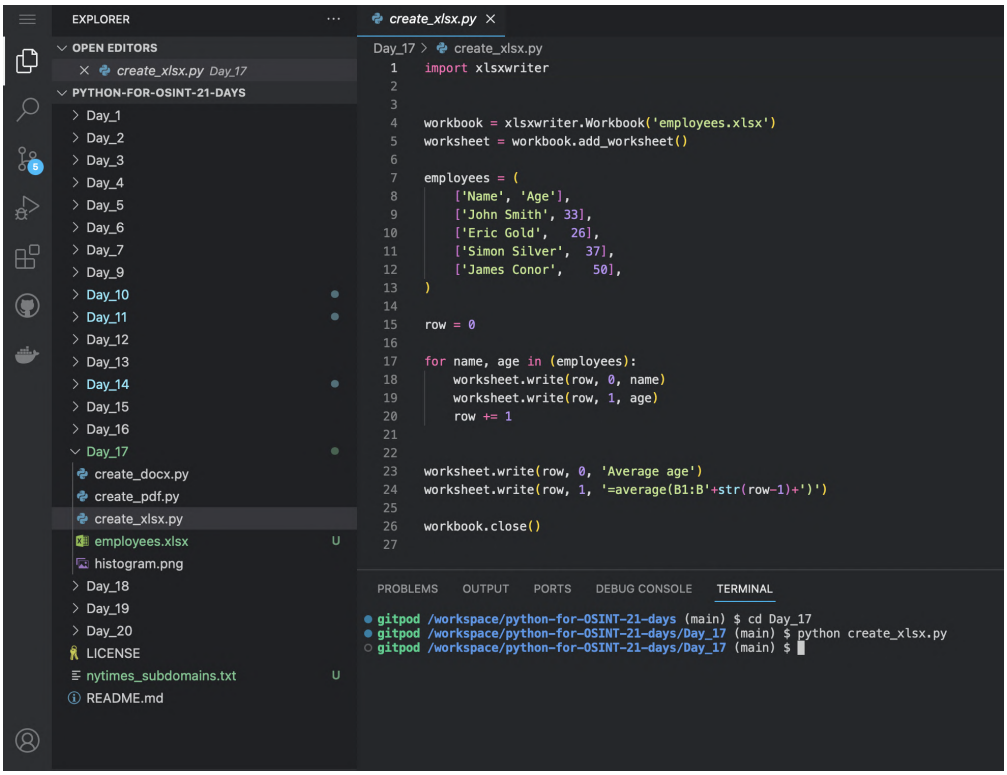
One important part of most research or investigation is the writing of the final report. In case you are creating many similar reports based on data with a similar structure, you can automate this process using Python. Let's try to create a simple MS Excel book.

```
gitpod /workspace/python-for-OSINT-21-days (main) $ pip install XlsxWriter
Collecting XlsxWriter
  Downloading XlsxWriter-3.1.0-py3-none-any.whl (152 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 152.7/152.7 kB 4.0 MB/s eta 0:00:00
Installing collected packages: XlsxWriter
Successfully installed XlsxWriter-3.1.0
```

Install the XlsxWriter package https://pypi.org/project/XlsxWriter/

```
pip install XlsxWriter
```

Run create_xlsx.py:




```
# Import the XlsxWriter package:
```

```
import xlsxwriter
```

Create a new file (book) named emploeyes.xlsx:

```
workbook = xlsxwriter.Workbook('employees.xlsx')
```

Add an empty sheet to the book:

```
worksheet = workbook.add_worksheet()
```

Create a two-dimensional list (employee name - age):

```
employees = (
  ['Name', 'Age'],
  ['John Smith', 33],
  ['Eric Gold',   26],
  ['Simon Silver',  37],
  ['James Conor',    50],
)
```

Go through the array and write the employee's name in column a and age in column B (creating a new row each time you try):

```
row = 0

for name, age in (employees):
  worksheet.write(row, 0, name)
  worksheet.write(row, 1, age)
  row += 1
```

At the lowest row, add the formula for calculating the average age:

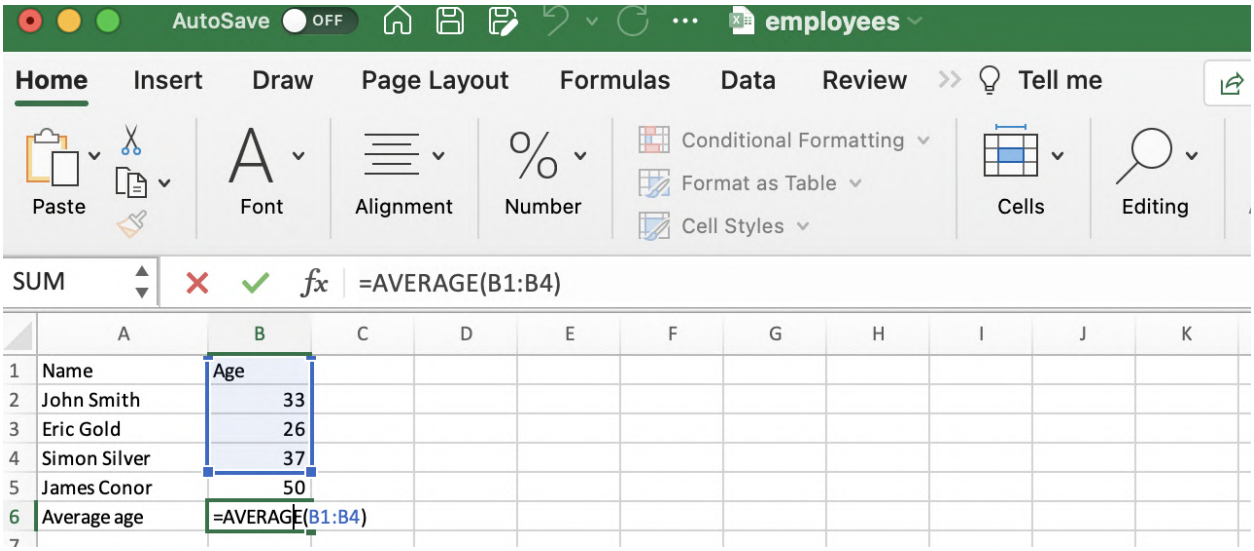
```
worksheet.write(row, 0, 'Average age')
```

```
worksheet.write(row, 1, '=average(B1:B'+str(row-1)+')')
```

Close the file:

```
workbook.close()
```

This is what the finished document would look like if you open it in Excel:

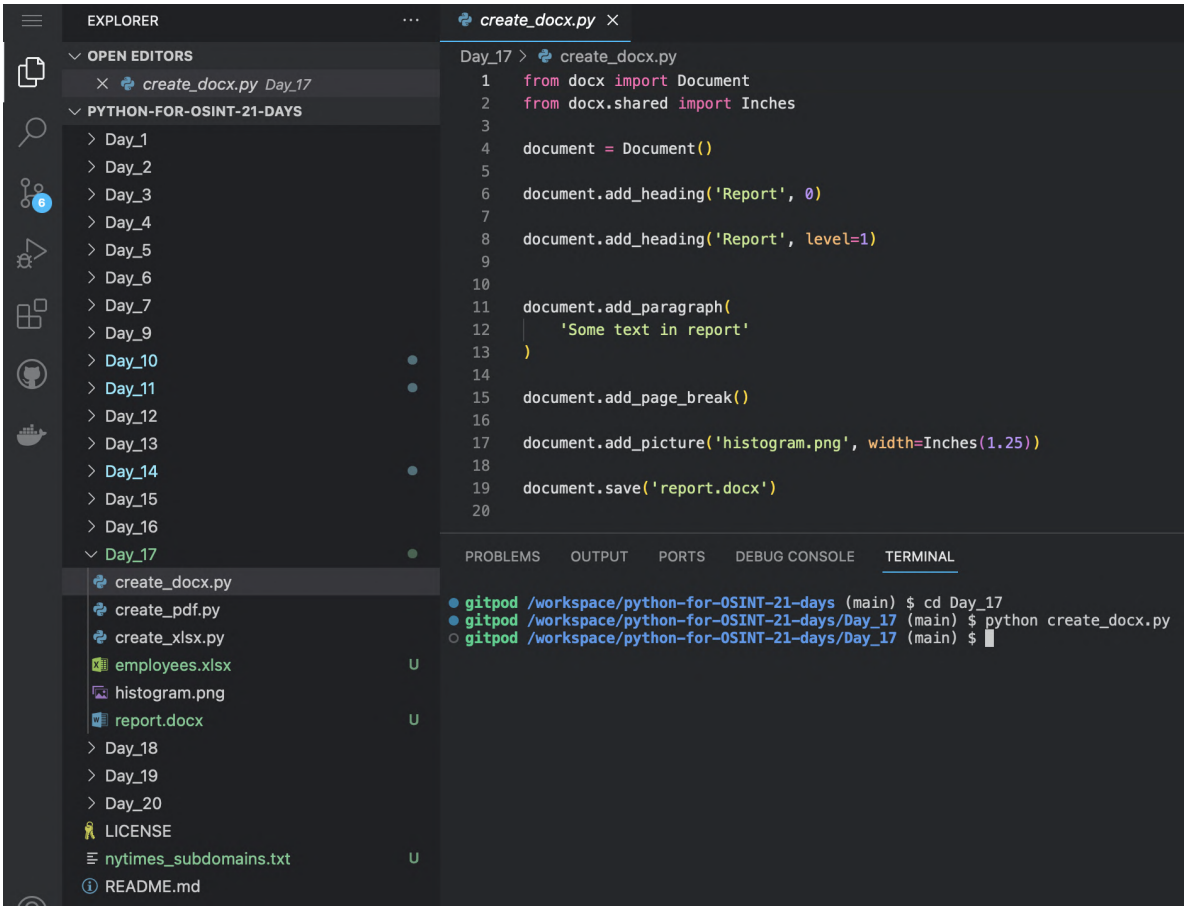


Now let's try to create a Word document. To do this we will use the python-docx package (https://python-docx.readthedocs.io/en/latest/).

```
gitpod /workspace/python-for-OSINT-21-days (main) $ pip install python-docx
Collecting python-docx
  Downloading python-docx-0.8.11.tar.gz (5.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.6/5.6 MB 21.3 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting lxml>=2.3.2
  Downloading lxml-4.9.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_24_x86_64.whl (7.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.2/7.2 MB 65.8 MB/s eta 0:00:00
Building wheels for collected packages: python-docx
  Building wheel for python-docx (setup.py) ... done
  Created wheel for python-docx: filename=python_docx-0.8.11-py3-none-any.whl size=184491 sha256=b7bc3f5e5dbc2e890532e6eaf57f7740702a
4517dfc7036d862f07944a88856e
  Stored in directory: /workspace/.pyenv_mirror/pip_cache/wheels/b2/11/b8/209e41af524253c9ba6c2a8b8ecec0f98ecbc28c732512803c
Successfully built python-docx
Installing collected packages: lxml, python-docx
Successfully installed lxml-4.9.2 python-docx-0.8.11
```

```
pip install python-docx
```

Run create_docx.py:



Importing packages:

```
from docx import Document
from docx.shared import Inches
```

Create a new document:

```
document = Document()
```

Add first title to the document:

```
document.add_heading('Report', 0)
```

Add a second title to the document:

```
document.add_heading('Report', level=1)
```

Add text paragraph to the document:

```
document.add_paragraph(  'Some text in report' )
```

Add a page break to the document:

```
document.add_page_break()
```

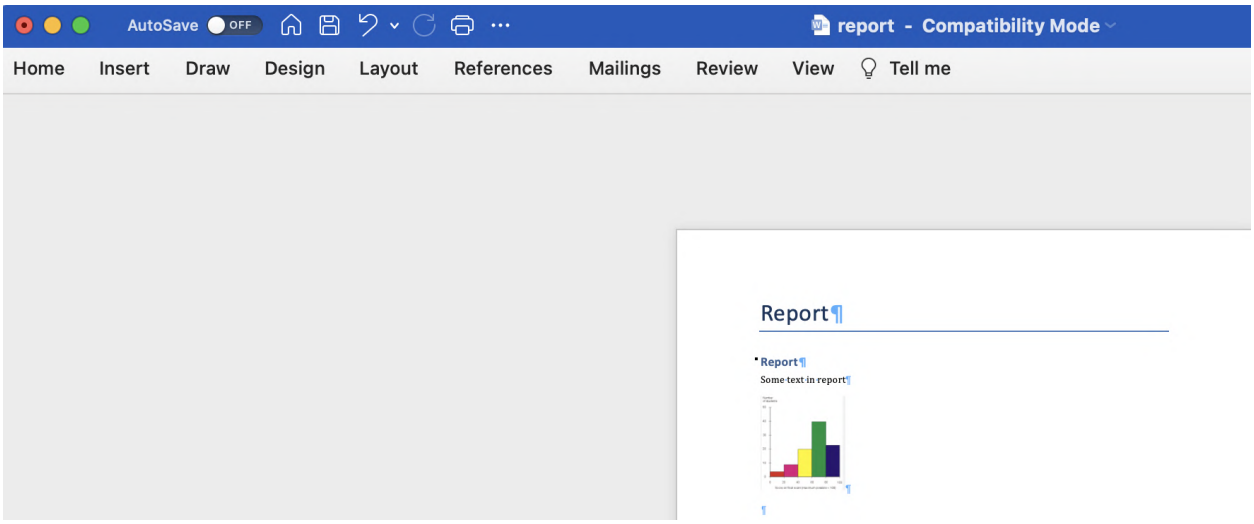
Add an image to the document:

```
document.add_picture('histogram.png', width=Inches(1.25))
```

Save the document:

```
document.save('report.docx')
```

This is what the resulting document will look like:

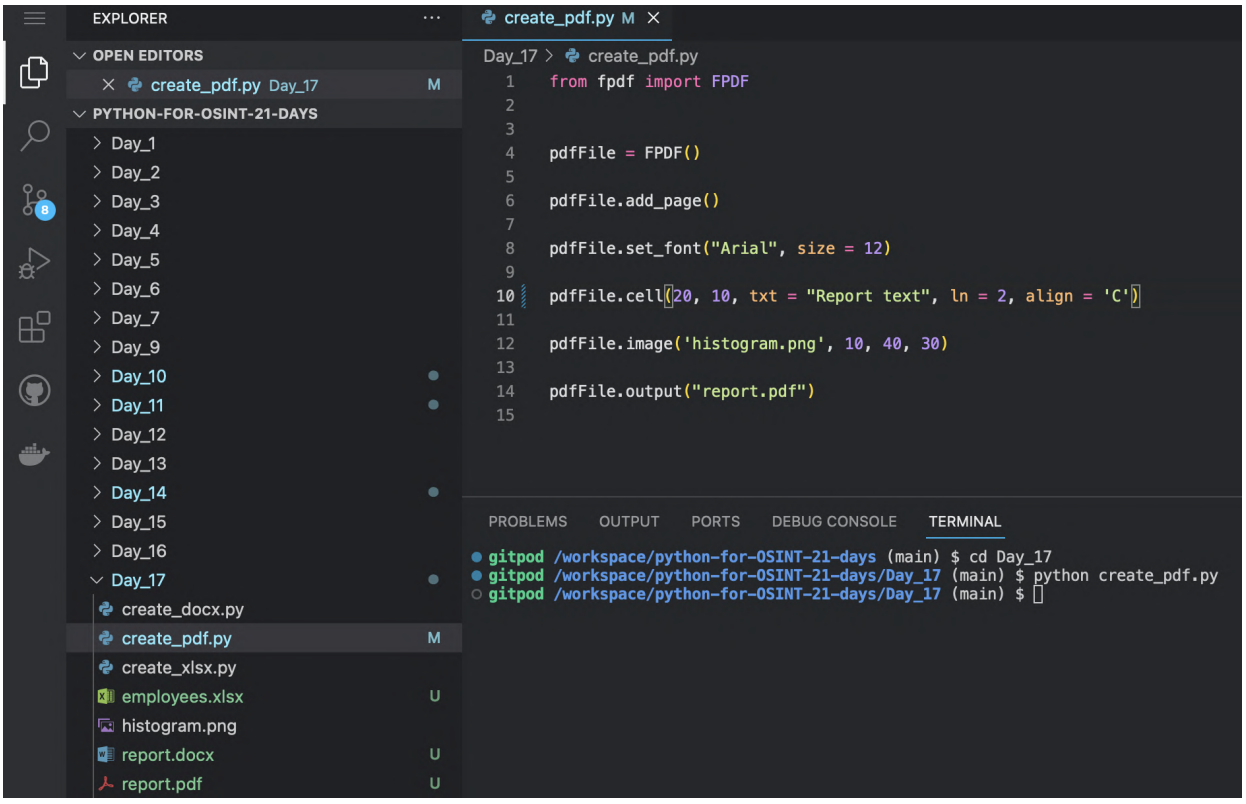


In the final part of our lesson, let's see how to generate PDF files. For this we will use FPDF (https://pyfpdf.readthedocs.io/en/latest/) packages.

```
gitpod /workspace/python-for-OSINT-21-days (main) $ pip install fpdf
Collecting fpdf
  Downloading fpdf-1.7.2.tar.gz (39 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: fpdf
  Building wheel for fpdf (setup.py) ... done
  Created wheel for fpdf: filename=fpdf-1.7.2-py2.py3-none-any.whl size=40704 sha256=ddf9fdad6eb46d4b13e3453e9c8f500c8d7fbf4e378c1e0d
17beea95e7bc9f4a
  Stored in directory: /workspace/.pyenv_mirror/pip_cache/wheels/65/4f/66/bbda9866da446a72e206d6484cd97381cbc7859a7068541c36
Successfully built fpdf
Installing collected packages: fpdf
Successfully installed fpdf-1.7.2
```

pip install fpdf

Run create_pdf.py:



Importing an FPDF package:

```
from fpdf import FPDF
```

Create an empty file:

```
pdfFile = FPDF()
```

Create a blank page in the file:

```
pdfFile.add_page()
```

Customize the document font:

```
pdfFile.set_font("Arial", size = 12)
```

Add text to the page, specifying the coordinates of the top and bottom indents:

```
pdfFile.cell(20, 10, txt = "Report text",   ln = 2, align = 'C')
```

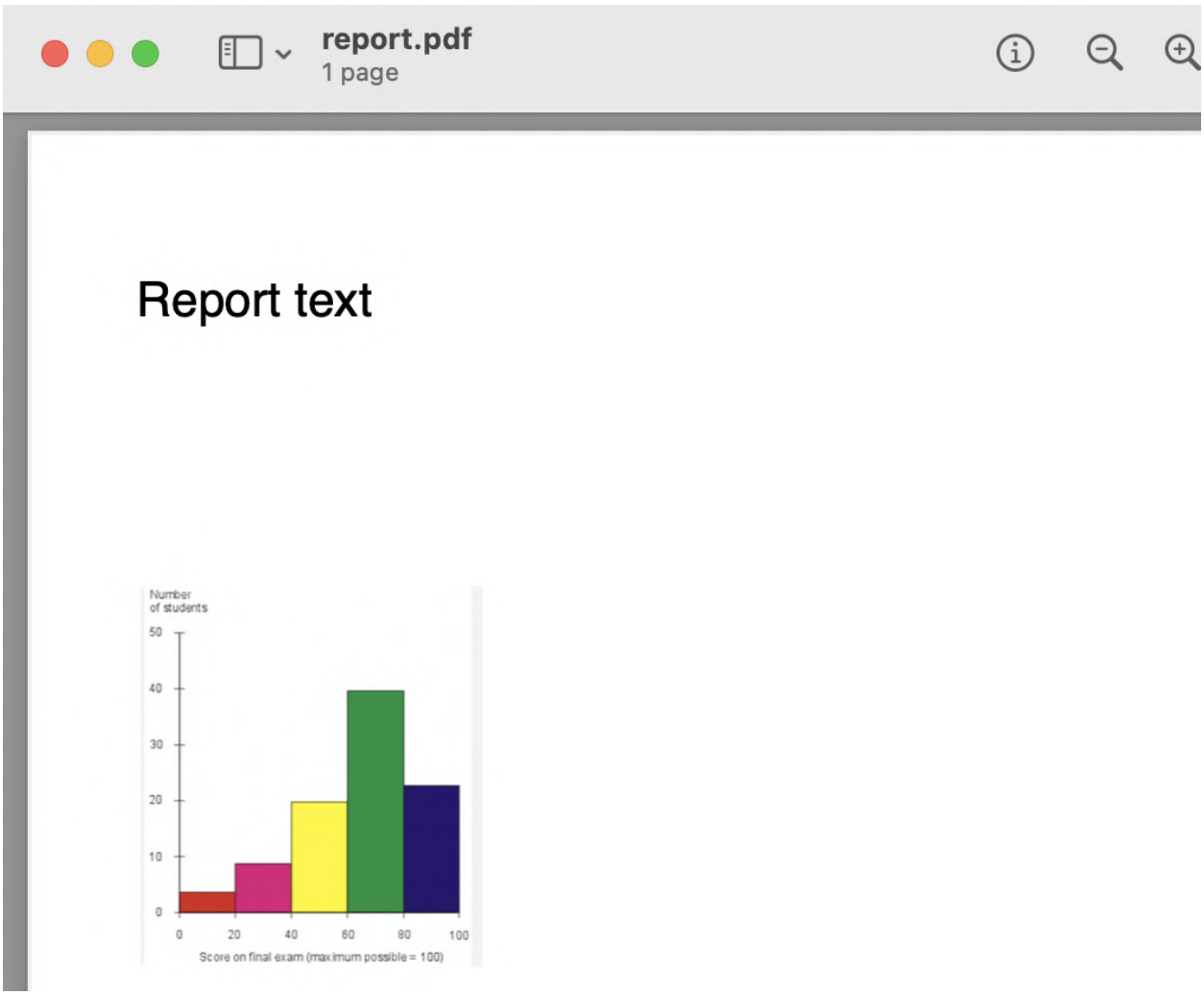
Add an image to the page, specifying its size and the coordinates of the top and bottom indents:

```
pdfFile.image('histogram.png', 10, 40, 30)
```

Save the result to the file report.pdf:

```
pdfFile.output("report.pdf")
```

This is what the result will look like if you open it in a PDF viewer:



Microsoft Power Point presentations can also be generated with Python using the package python-pptx (https://python-pptx.readthedocs.io/en/latest/).

Day 18. Generating charts and maps

In the last lesson, we inserted a picture of a diagram into a document. Such pictures can also be generated automatically, based on the data obtained during the investigation. For example, you can visualize data from different APIs

Matplotlib (https://matplotlib.org) will help us with this. This is a library for creating data visualizations. We will also use the NumPy package (https://numpy.org), which will help us work with multi-dimensional arrays.

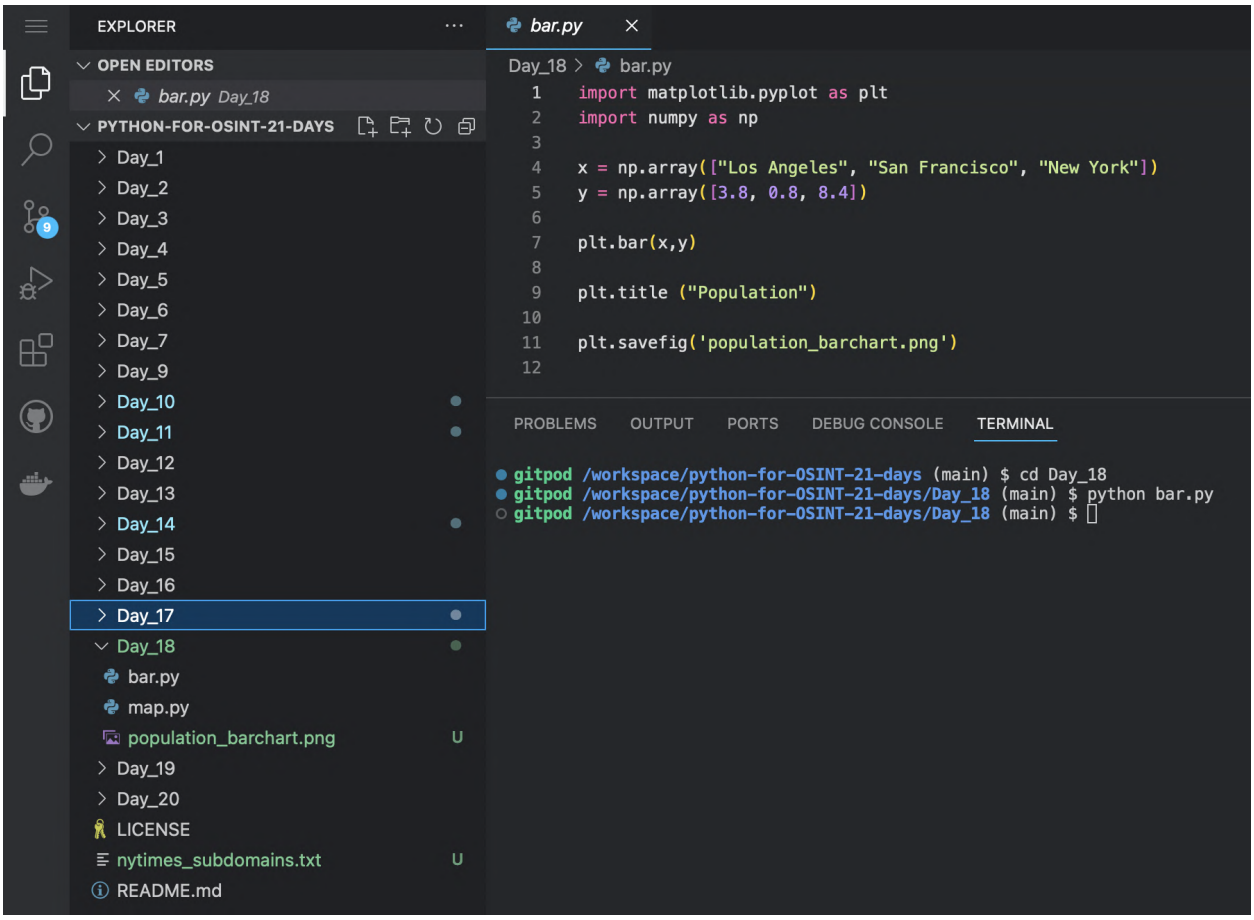
Generating diagrams

```
gitpod /workspace/python-for-OSINT-21-days (main) $ pip install matplotlib
Collecting matplotlib
  Downloading matplotlib-3.7.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.6/11.6 MB 43.6 MB/s eta 0:00:00
Collecting contourpy>=1.0.1
  Downloading contourpy-1.0.7-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (299 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 300.0/300.0 kB 43.7 MB/s eta 0:00:00
Collecting cycler>=0.10
  Downloading cycler-0.11.0-py3-none-any.whl (6.4 kB)
Collecting fonttools>=4.22.0
  Downloading fonttools-4.39.3-py3-none-any.whl (1.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.0/1.0 MB 61.5 MB/s eta 0:00:00
Collecting kiwisolver>=1.0.1
  Downloading kiwisolver-1.4.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.4/1.4 MB 70.0 MB/s eta 0:00:00
Collecting numpy>=1.20
  Downloading numpy-1.24.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (17.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.3/17.3 MB 83.5 MB/s eta 0:00:00
Requirement already satisfied: packaging>=20.0 in /home/gitpod/.pyenv/versions/3.11.1/lib/python3.11/site-packages (from matplotlib)
(23.0)
Collecting pillow>=6.2.0
  Downloading Pillow-9.5.0-cp311-cp311-manylinux_2_28_x86_64.whl (3.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.4/3.4 MB 79.5 MB/s eta 0:00:00
Collecting pyparsing>=2.3.1
  Downloading pyparsing-3.0.9-py3-none-any.whl (98 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.3/98.3 kB 19.0 MB/s eta 0:00:00
Requirement already satisfied: python-dateutil>=2.7 in /home/gitpod/.pyenv/versions/3.11.1/lib/python3.11/site-packages (from matplot
lib) (2.8.2)
Requirement already satisfied: six>=1.5 in /home/gitpod/.pyenv/versions/3.11.1/lib/python3.11/site-packages (from python-dateutil>=2.
7->matplotlib) (1.16.0)
Installing collected packages: pyparsing, pillow, numpy, kiwisolver, fonttools, cycler, contourpy, matplotlib
Successfully installed contourpy-1.0.7 cycler-0.11.0 fonttools-4.39.3 kiwisolver-1.4.4 matplotlib-3.7.1 numpy-1.24.3 pillow-9.5.0 pyp
arsing-3.0.9
```

Install Matplotlib before running script (numpy is installed automatically when you install Matplotlib):

```
pip install matplotlib
```

Run bar.py:



Importing matplotlib and numpy packages:

```
import matplotlib.pyplot as plt
import numpy as np
```

Create a list with the names of the cities:

```
x = np.array(["Los Angeles", "San Francisco", "New York"])
```

Create a list with numerical values of their population in millions:

```
y = np.array([3.8, 0.8, 8.4])
```

Plot a graph on the basis of these two lists:

```
plt.bar(x,y)
```

Add a title to the chart:

```
plt.title ("Population")
```

Save result in a file:

```
plt.savefig('population_barchart.png')
```

In this example, we used the NumPy library to create two one-dimensional arrays so that we could then make a two-dimensional array from which to plot the graph.

In the last lesson we created two-dimensional lists without using numpy, but if you are going to use lists to create any visualizations using Matplotlib, it is better to use numpy arrays (once again, note that lists and arrays in Python are different things).

Matplotlib allows you to create many different types of data visualizations: line chart, scatter chart, step chart, pie chart and hundreds of others. And in combination with Basemap toolkit (https://matplotlib.org/basemap/), you can even visualize geodata.

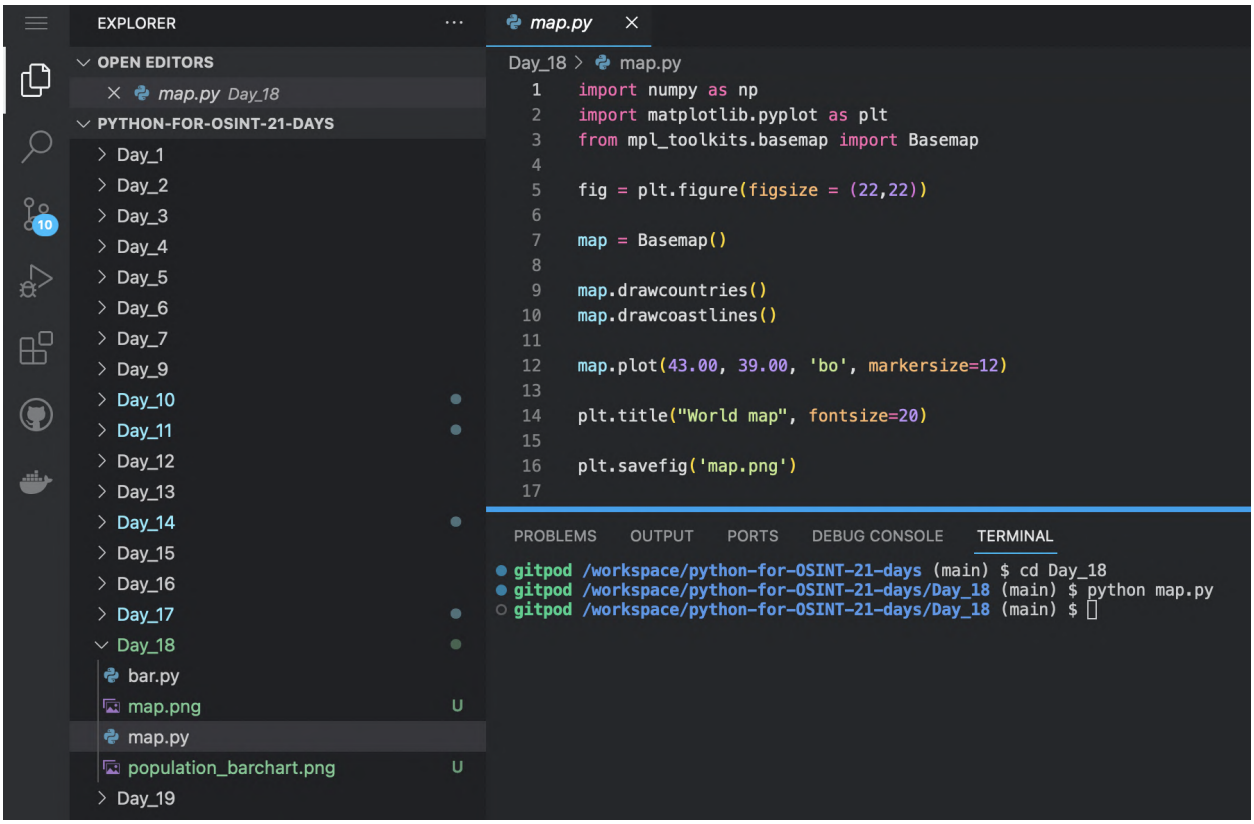
Generating maps

```
gitpod /workspace/python-for-OSINT-21-days (main) $ pip install basemap
Collecting basemap
  Downloading basemap-1.3.6-cp311-cp311-manylinux1_x86_64.whl (860 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 860.3/860.3 kB 8.0 MB/s eta 0:00:00
Collecting basemap-data<1.4,>=1.3.2
  Downloading basemap_data-1.3.2-py2.py3-none-any.whl (30.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 30.5/30.5 MB 62.6 MB/s eta 0:00:00
Collecting pyshp<2.4,>=1.2
  Downloading pyshp-2.3.1-py2.py3-none-any.whl (46 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 46.5/46.5 kB 11.6 MB/s eta 0:00:00
Collecting matplotlib<3.7,>=1.5
  Downloading matplotlib-3.6.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.8/11.8 MB 112.1 MB/s eta 0:00:00
Collecting pyproj<3.5.0,>=1.9.3
  Downloading pyproj-3.4.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (7.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.8/7.8 MB 115.6 MB/s eta 0:00:00
Collecting numpy<1.24,>=1.22
  Downloading numpy-1.23.5-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (17.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.1/17.1 MB 106.6 MB/s eta 0:00:00
Requirement already satisfied: contourpy>=1.0.1 in /workspace/.pyenv_mirror/user/current/lib/python3.11/site-packages (from matplotli
b<3.7,>=1.5->basemap) (1.0.7)
Requirement already satisfied: cycler>=0.10 in /workspace/.pyenv_mirror/user/current/lib/python3.11/site-packages (from matplotlib<3.
7,>=1.5->basemap) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /workspace/.pyenv_mirror/user/current/lib/python3.11/site-packages (from matplotl
ib<3.7,>=1.5->basemap) (4.39.3)
Requirement already satisfied: kiwisolver>=1.0.1 in /workspace/.pyenv_mirror/user/current/lib/python3.11/site-packages (from matplotl
ib<3.7,>=1.5->basemap) (1.4.4)
Requirement already satisfied: packaging>=20.0 in /home/gitpod/.pyenv/versions/3.11.1/lib/python3.11/site-packages (from matplotlib<3
.7,>=1.5->basemap) (23.0)
```

Install the basemap package before running script:

```
pip install basemap
```

Run map.py:



```
# Importing matplotlib, nump, basemap packages:

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap

# Create an empty image:

fig = plt.figure(figsize = (22,22))

# Create a map (by default it creates a world map):

map = Basemap()

# Add country boundaries to it:

map.drawcountries()

# Add coastlines to it:

map.drawcoastlines()

# Add a point to it, specifying geographic coordinates, marker type and
size:

map.plot(43.00, 39.00, 'bo', markersize=12)

# Add title to the map:

plt.title("World map", fontsize=20)

# Save result to a file:

plt.savefig('map.png')
```

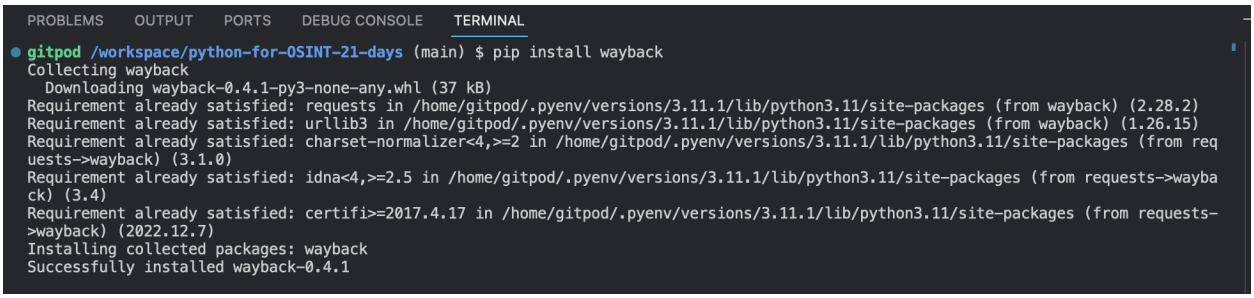
This is what the resulting map.png file should look like:

Basemap allows you to load any maps from shape files (with shp extension). This allows you to make visualizations for individual countries and regions. You can read more about it here: https://basemaptutorial.readthedocs.io/en/latest/shapefile.html.

Day 19. Wayback Machine and time/date functions

Archive.org is one of the most important sources of information at OSINT and SOCMINT (Social media intelligence). It allows you to find deleted contact information of site owners, trace the history of information changes in the profile in the social network or find any other deleted content.

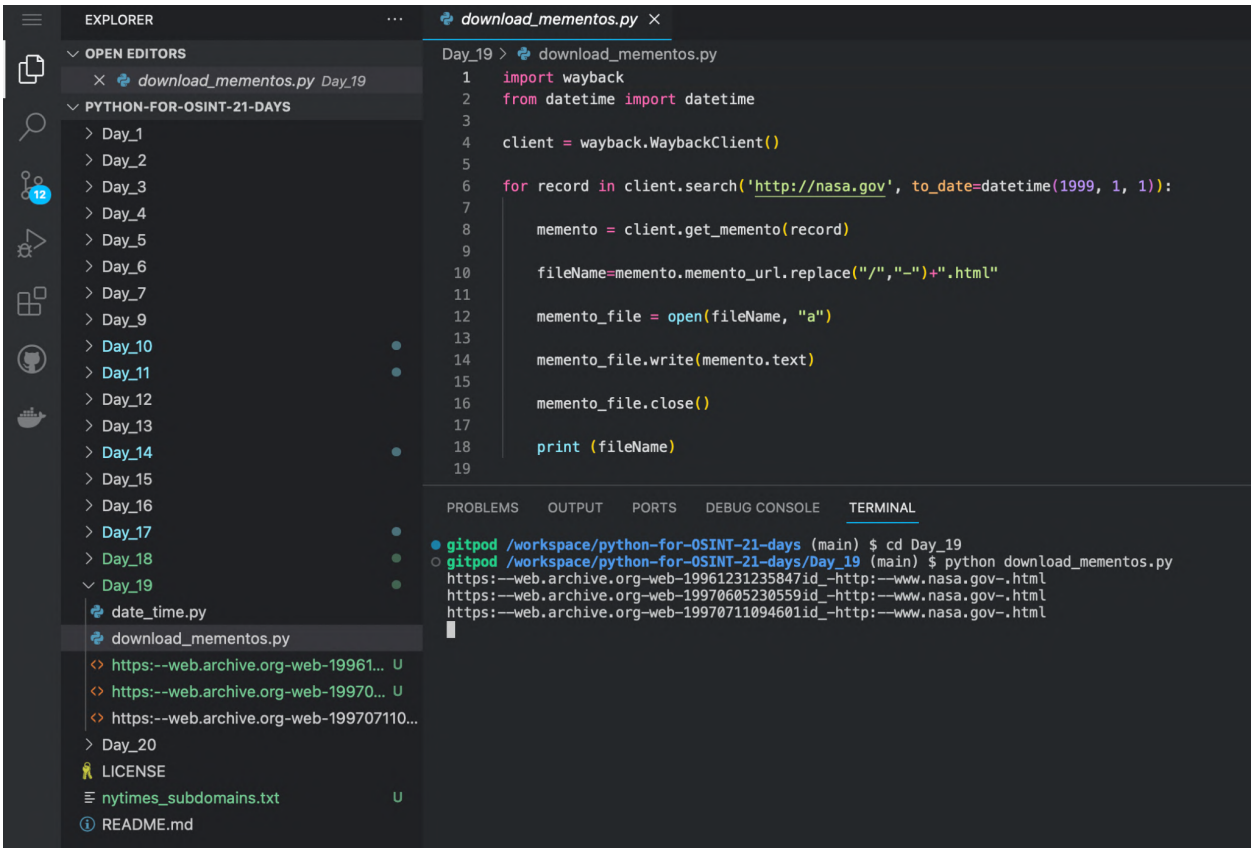
We will use Wayback package (https://pypi.org/project/wayback/) for automation work with WaybackMachine. Also, working with archives is a great excuse to finally learn the basics of working with date and time in Python.



Install wayback package before running script:

```
pip install wayback
```

Run download_mementos.py:



Importing pathback and datetime packages:

```
import wayback
from datetime import datetime
```

Create web archive client:

```
client = wayback.WaybackClient()
```

Search for copies of nasa.gov web pages saved before 1999:

```
for record in client.search('http://nasa.gov', to_date=datetime(1999, 1, 1)):
```

Get memento record (web page copy):

```
  memento = client.get_memento(record)
```

Generate the name of the file in which we will save the HTML code of the web page copy (replace the link to the page with / to - (so that no error occurs when saving the file) and add .html extension:

```
  fileName=memento.memento_url.replace("/","-")+".html"
```

Open file in which we will save the HTML code of the web page copy:

```
  memento_file = open(fileName, "a")
```

Write HTML code of the web page copy to the file:

```
  memento_file.write(memento.text)
```

Close file:

```
  memento_file.close()
```

Print name of file:

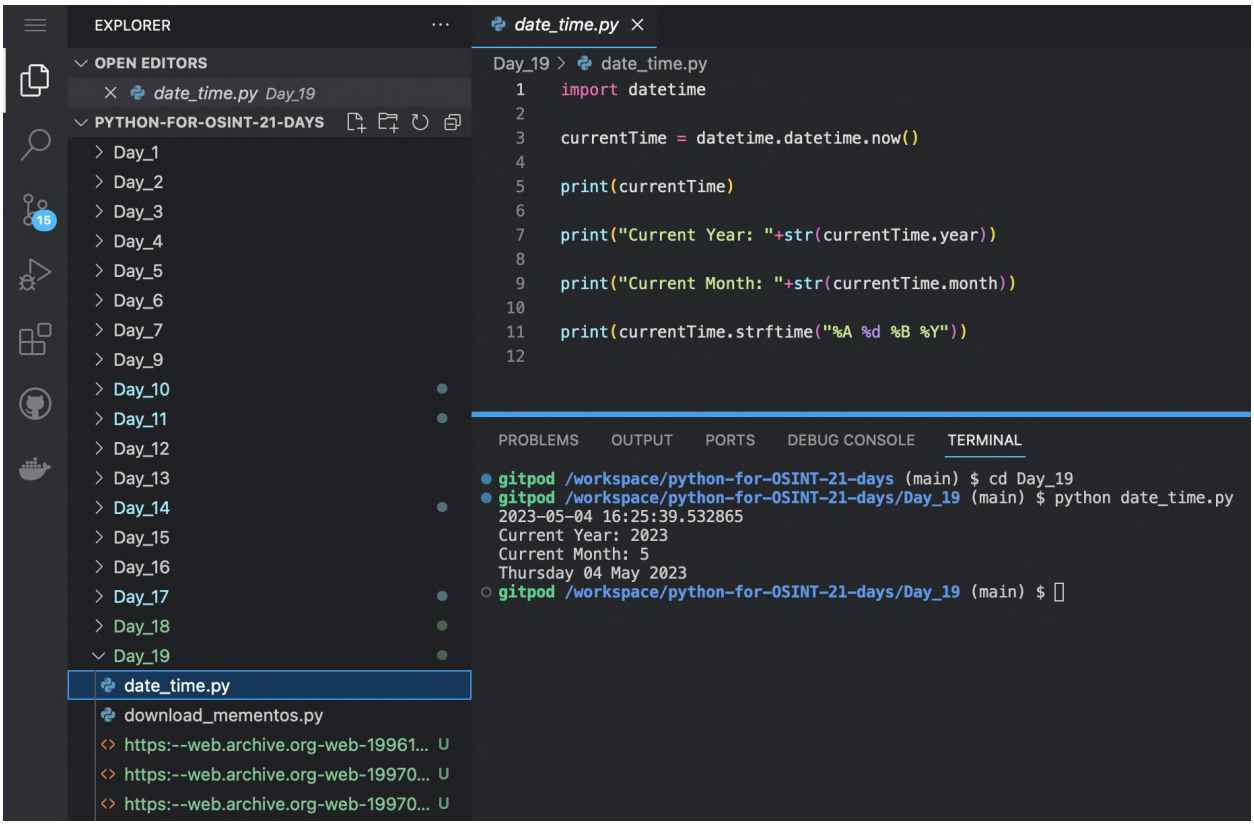
```
  print (fileName)
```

Be prepared for the fact that the script may take some time to run.

Note that in the code above, we used datetime() to set the date range for finding copies of the web page in the web archive.

This is a very important function. Let's look at some examples of how to work with it.

Run date_time.py:



Import the datetime package (available in Python by default):

```
import datetime
```

Put the current date and time into a variable:

```
currentTime = datetime.datetime.now()
```

Display the current date and time:

```
print(currentTime)
```

Display the current year:

```
print("Current Year: "+str(currentTime.year))
```

Display the current month:

```
print("Current Month: "+str(currentTime.month))
```

Displays current day of the week, day of the month, month and year:

```
print(currentTime.strftime("%A %d %B %Y"))
```

You can replace datetime.datetime.now() to other date. For example, datetime.datetime(2023, 5, 4).

Day 20. Web apps creation

After posts about any OSINT command line tools, readers sometimes ask me, "Isn't there a web version of this tool?"

After taking this course, you are unlikely to have serious difficulty using the command line tools. After all, you now know how easy it is.

But nevertheless, many people have them.

And if you've made some useful Python script and you want as many people as possible to use it, you should consider turning it into a web application.

There are many ways to create web applications in Python. For example, you can use frameworks such as Django, Flask, Dash, Falcon etc. They are fairly easy to use and learn, but still, for this course, I chose the easiest and fastest option - Streamlit package (https://streamlit.io).

Please install it using pip:

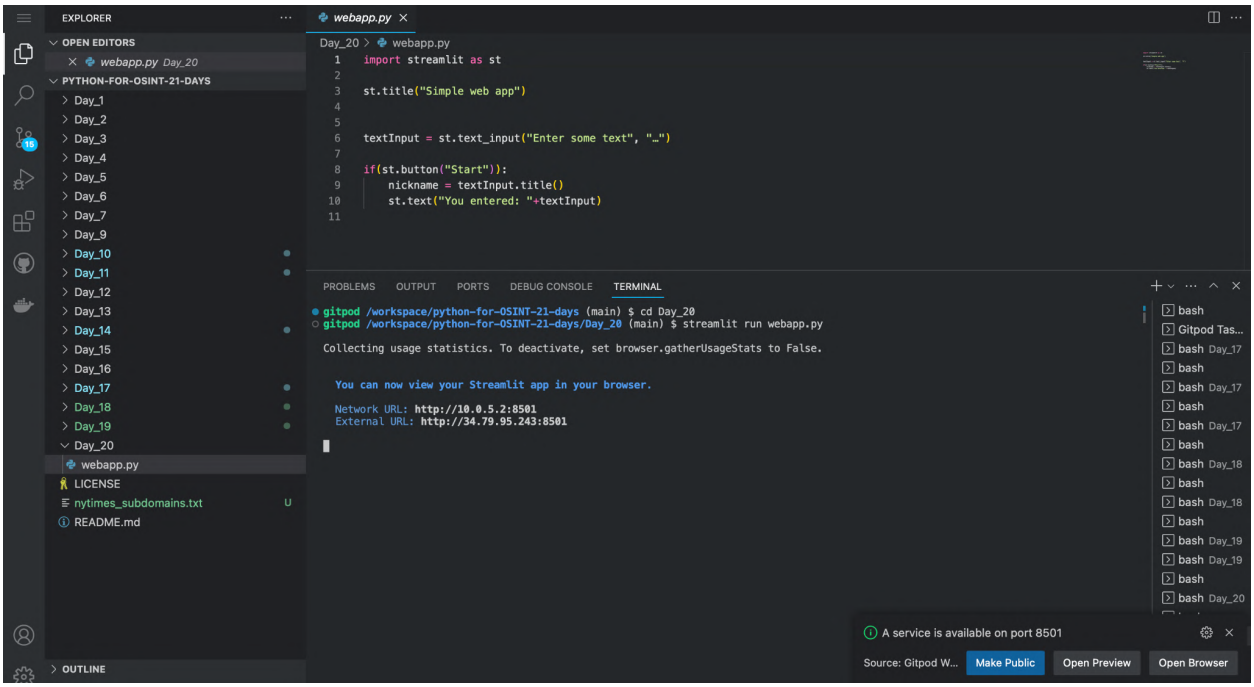
```
gitpod /workspace/python-for-OSINT-21-days (main) $ pip install streamlit
Collecting streamlit
  Downloading streamlit-1.22.0-py2.py3-none-any.whl (8.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.9/8.9 MB 28.2 MB/s eta 0:00:00
Collecting altair<5,>=3.2.0
  Downloading altair-4.2.2-py3-none-any.whl (813 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 813.6/813.6 kB 57.9 MB/s eta 0:00:00
Collecting blinker>=1.0.0
  Downloading blinker-1.6.2-py3-none-any.whl (13 kB)
Collecting cachetools>=4.0
  Downloading cachetools-5.3.0-py3-none-any.whl (9.3 kB)
Requirement already satisfied: click>=7.0 in /workspace/.pyenv_mirror/user/current/lib/python3.11/site-packages (from streamlit) (8.1
.3)
Requirement already satisfied: importlib-metadata>=1.4 in /home/gitpod/.pyenv/versions/3.11.1/lib/python3.11/site-packages (from stre
amlit) (6.1.0)
Requirement already satisfied: numpy in /workspace/.pyenv_mirror/user/current/lib/python3.11/site-packages (from streamlit) (1.23.5)
Requirement already satisfied: packaging>=14.1 in /home/gitpod/.pyenv/versions/3.11.1/lib/python3.11/site-packages (from streamlit) (
23.0)
Collecting pandas<3,>=0.25
  Downloading pandas-2.0.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.2/12.2 MB 86.6 MB/s eta 0:00:00
Requirement already satisfied: pillow>=6.2.0 in /workspace/.pyenv_mirror/user/current/lib/python3.11/site-packages (from streamlit) (
9.5.0)
Collecting protobuf<4,>=3.12
  Downloading protobuf-3.20.3-py2.py3-none-any.whl (162 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 162.1/162.1 kB 35.4 MB/s eta 0:00:00
Collecting pyarrow>=4.0
  Downloading pyarrow-12.0.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (38.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 38.9/38.9 MB 56.2 MB/s eta 0:00:00
Collecting pympler>=0.9
  Downloading Pympler-1.0.1-py3-none-any.whl (164 kB)
```

pip install streamlit

Now let's launch our first web application:

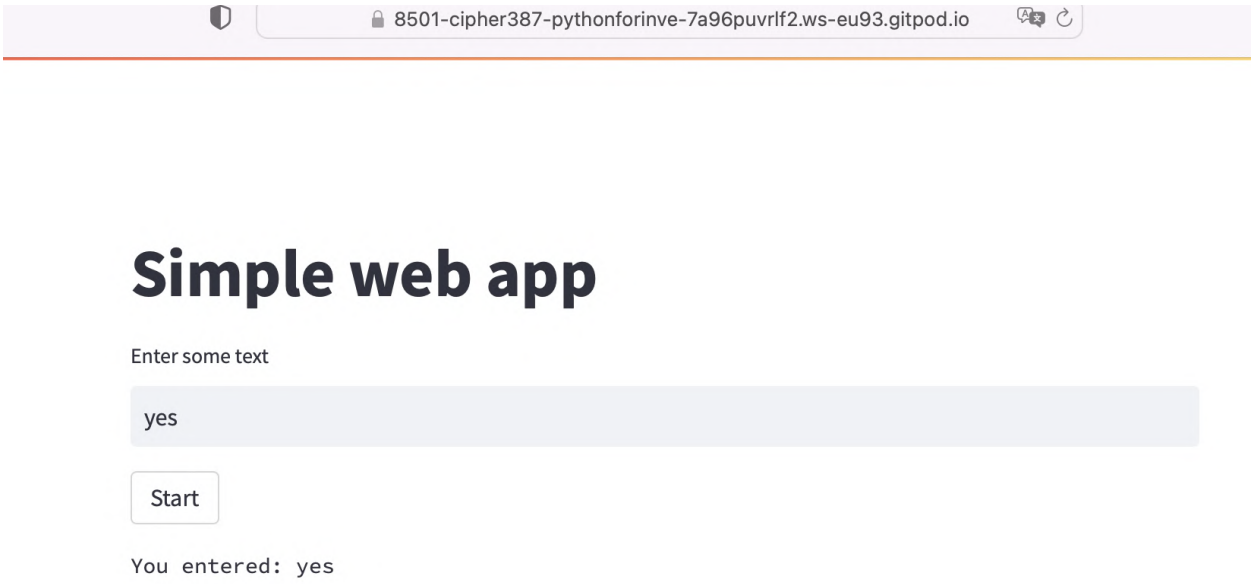
Run webapp.py (note that we do this in a different way than we did with the other files in this tutorial):

```
streamlit run webapp.py
```



After launching, copy the Network URL and open it in a browser.

If you use Gitpod, just click Open Browser.



The result should be a simple web application that displays the text entered by the user after clicking on the Start button.

Let's take a look at the application code.

Importing Streamlit package:

```
import streamlit as st
```

Create a header of the web page:

```
st.title("Simple web app")
```

Create a text box:

```
textInput = st.text_input("Enter some text", "…")
```

Create the button, on pressing which the text entered in the text field will be displayed:

```
if(st.button("Start")):
```

```
  nickname = textInput.title()
  st.text("You entered: "+textInput)
```

At the end of this lesson I suggest you to read an article in which I tell you how to use Streamlit to turn Maigret (tool for nickname enumeration) into a web application:

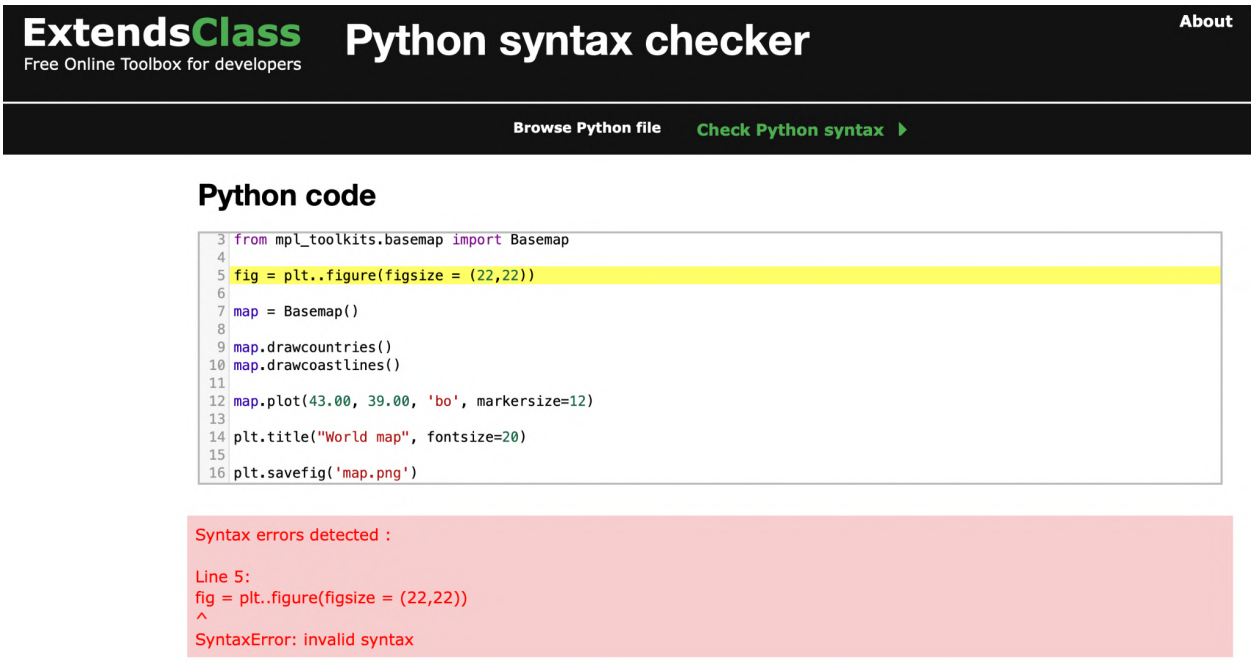
The easiest way to turn an OSINT Python script into a web application. Combining Maigret and Streamlit in three simple steps

From it you will learn more about the various useful features of Streamlit.

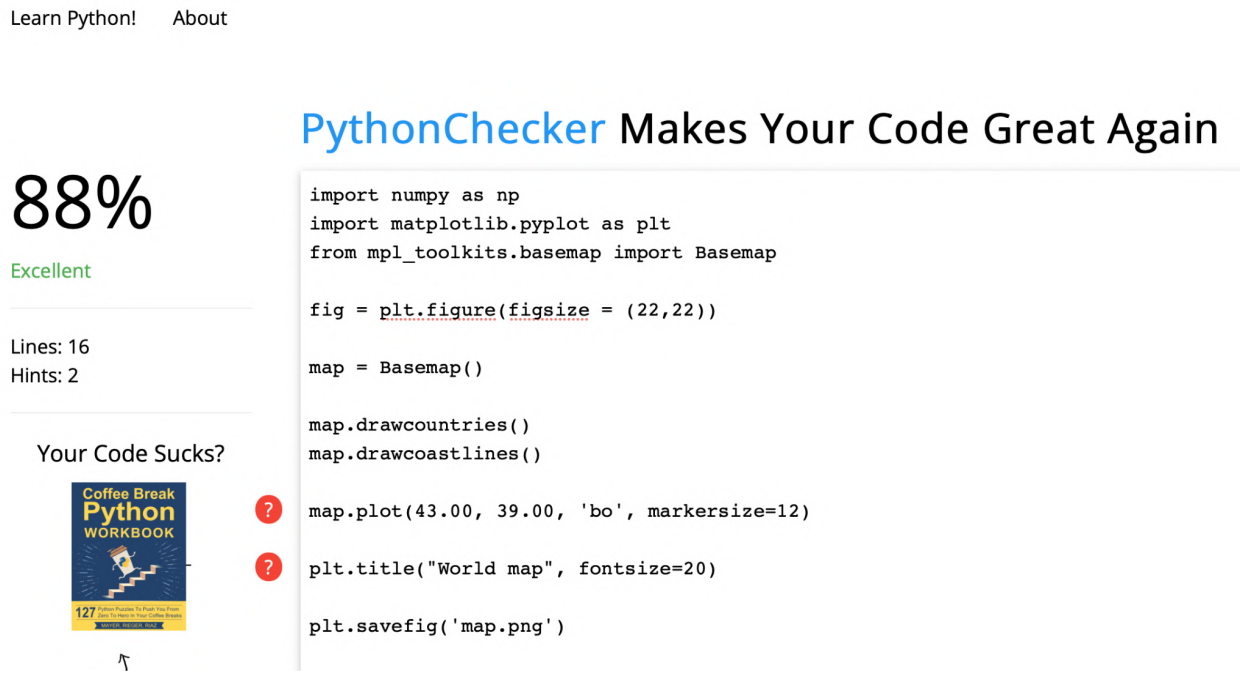
Day 21. Tools to help you work with code

If you write your own Python code after finishing this course, you will probably run into different problems all the time.

Firstly, you may occasionally have scripts that don't run because of syntax errors.

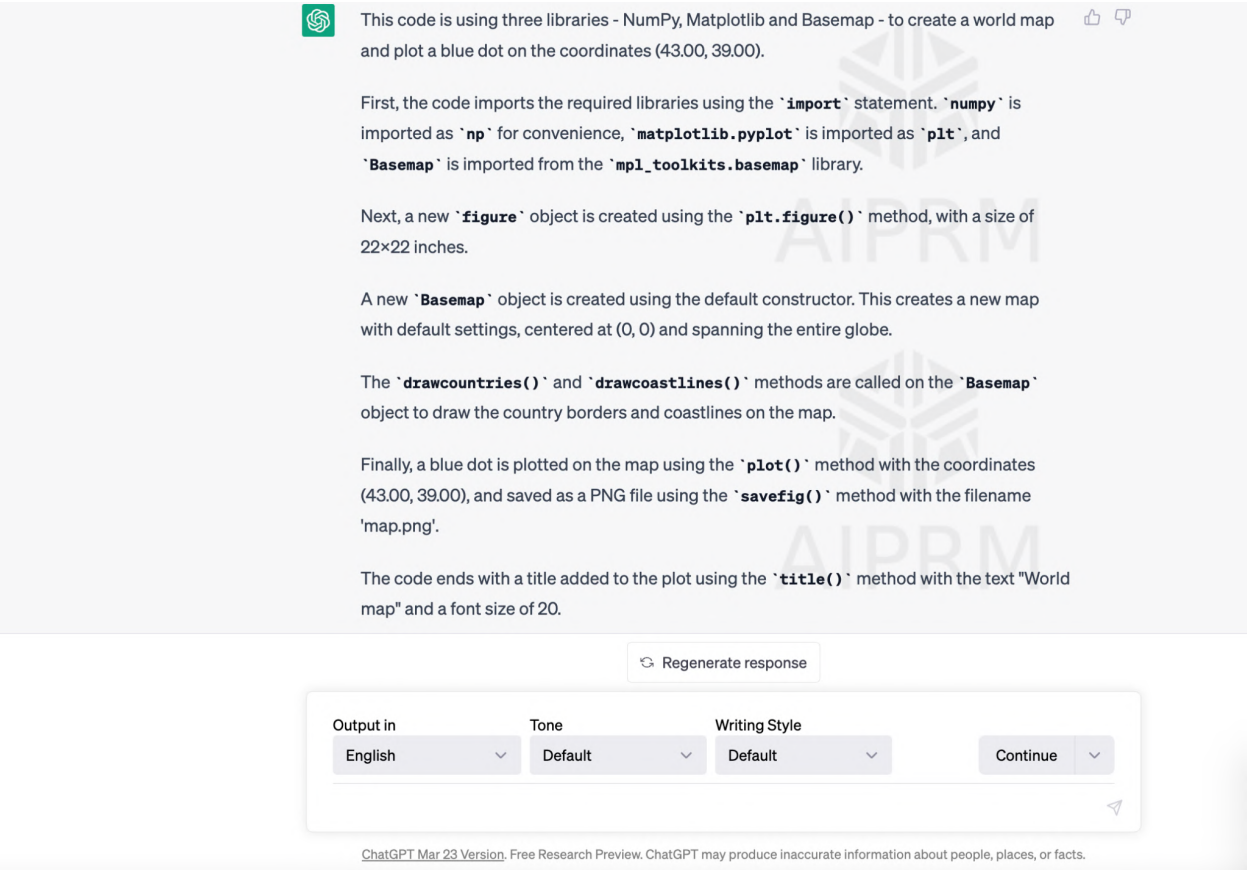


But fortunately, they can be found very quickly using numerous online syntax-checking services. For example, https://extendsclass.com/python-tester.html.



If you want to show your code to someone else and want to make it as easy to read as possible, I recommend using https://www.pythonchecker.com.

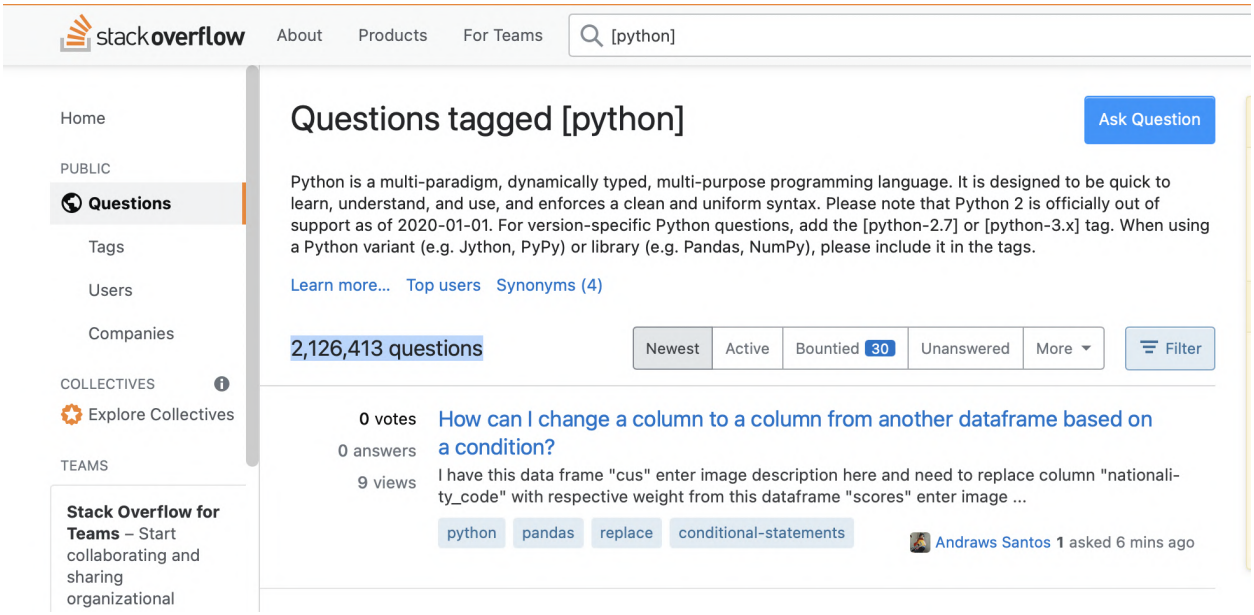
It will tell you where to put extra spaces and line breaks, and point out comments that are too long in the code. In other words, it will find problems that do not hinder the execution of the code, but make it less understandable for other developers.



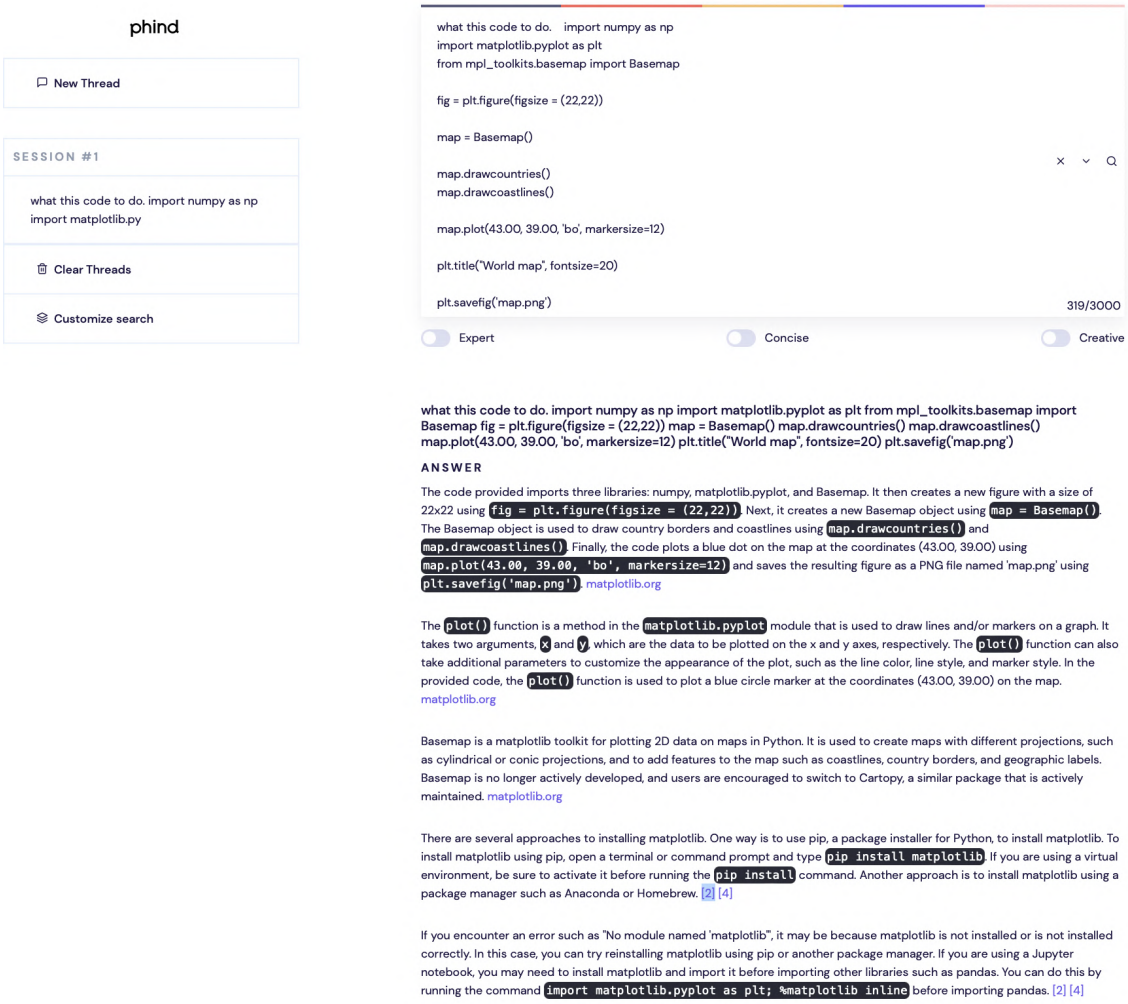
And, of course, you could just ask ChatGPT to refine your code and suggest solutions to any problems. It will also help you make sense of other people's code.

But the most important thing to remember when working with ChatGPT is that it makes VERY many mistakes.

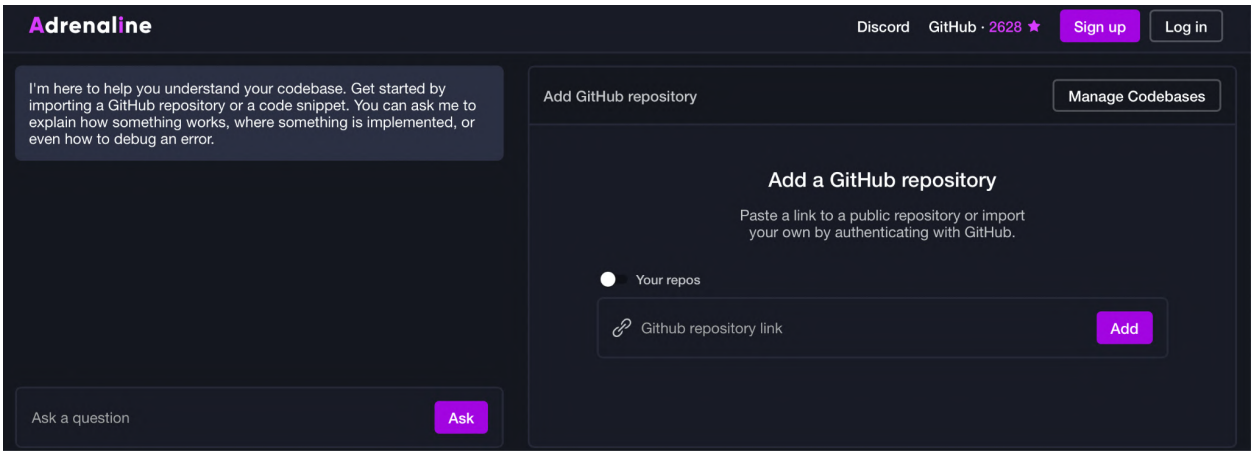
Therefore, it is better to strive to figure out solutions to different problems on your own, so that you understand all the details.



Stackoverflow is the world's largest online community where you can discuss issues related to programming and computer technology in general. There are over 2 million Python-related questions already created and you can really find solutions to most problems.



Phind.com is an AI search tool that searches for relevant answers on Stackoverflow, analyzes them and generates a solution based on them. Sometimes he also uses other IT sites as sources of information.



Sometimes, in order to find the code that performs a particular task in a larger project, you have to go through a huge Github repository of many related files.

https://useadrenaline.com/app will help to automate this work. Simply add a link to one or more repositories, and then ask questions about their content to AI chatbot.

More tools to make the code easier to work with can be found in this repository on my Github profile:

https://github.com/cipher387/code-understanding-tools

What to do next

It depends on what your goals are. If you are an OSINT specialist, then I recommend you to practice more investigations and try to use some tools written in Python (there are a lot of them in my Twitter account), and to get additional knowledge as needed when you face new tasks.

If your goal is to develop some serious project in Python, you definitely need additional courses. For example, you can start by taking the free official course https://www.learnpython.org/ from Python.org and Datacamp.

It's likely that after completing this course, you've had ideas for some new simple tools for OSINT. You already have enough knowledge to do a lot of useful things.

If you want to make your tool public and find your users, I recommend reading this article:

10 very simple tips for OSINT tools developers

And don't forget to subscribe to me on Twitter, Mastodon, Telegram, Substack, Discord, Medium, Substack to get regular news about new OSINT tools, and not to miss new training courses.

Table of contents