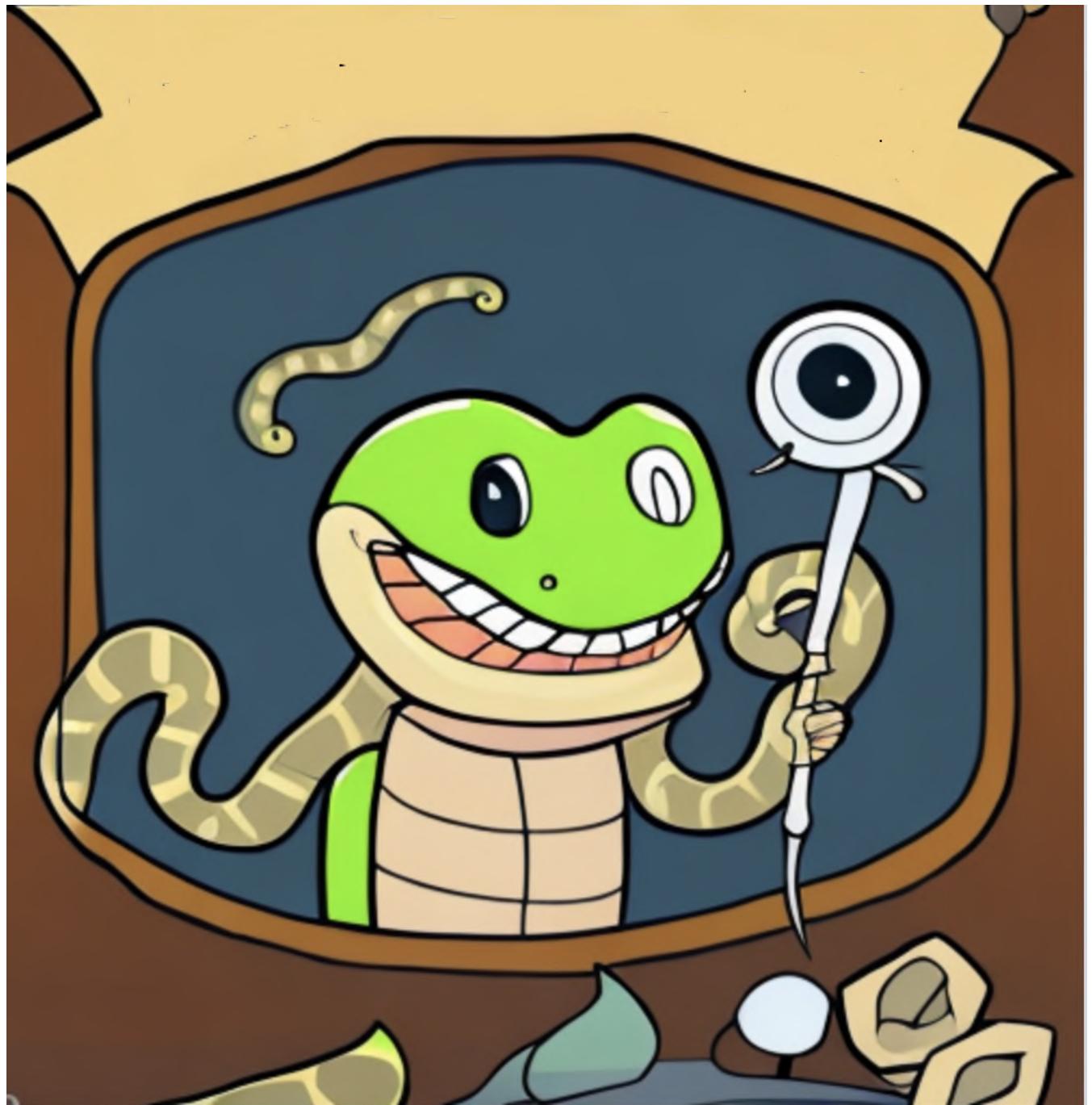


Python for OSINT. 21-day course for beginners



[@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.

```
PROBLEMS    OUTPUT    PORTS    DEBUG CONSOLE    TERMINAL

gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_18
gitpod /workspace/python-for-OSINT-21-days/Day_18 (main) $ python map.py
Traceback (most recent call last):
  File "/workspace/python-for-OSINT-21-days/Day_18/map.py", line 1, in <module>
    import numpy as np
ModuleNotFoundError: No module named 'numpy'
gitpod /workspace/python-for-OSINT-21-days/Day_18 (main) $ █
```

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

For example:

```
pip install numpy
```

```
gitpod /workspace/python-for-OSINT-21-days/Day_18 (main) $ pip install numpy
Collecting numpy
  Downloading numpy-1.24.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (17.3 MB)
    17.3/17.3 MB 21.1 MB/s eta 0:00:00
Installing collected packages: numpy
Successfully installed numpy-1.24.3
```

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
```

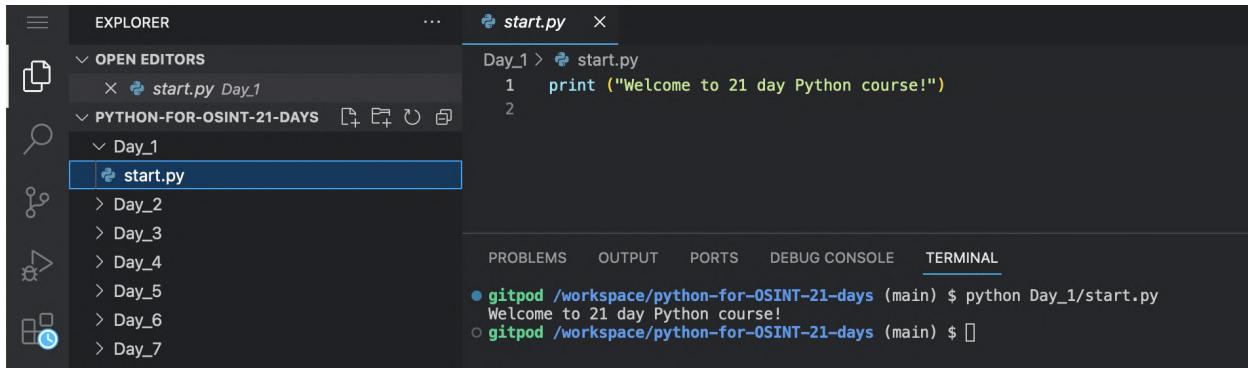
If you see a message asking for your username and password, enter your Github account username and password.

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:



```
start.py
Day_1 > start.py
1   print ("Welcome to 21 day Python course!")
2

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL
● gitpod /workspace/python-for-OSINT-21-DAYS (main) $ python Day_1/start.py
  Welcome to 21 day Python course!
○ gitpod /workspace/python-for-OSINT-21-DAYS (main) $ 
```

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`

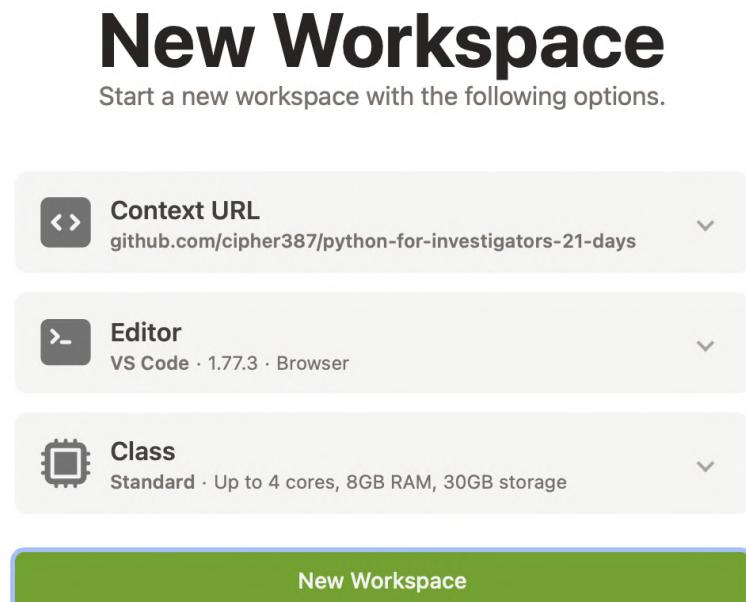
If the folder “python-for-OSINT-21-days” is not copied, check if you entered the password correctly.

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.

The screenshot shows the Visual Studio Code interface. The Explorer pane on the left displays a file tree with a folder named 'Day_1' containing a file 'start.py'. The Terminal pane on the right shows the output of running the script: 'Welcome to 21 day Python course!'. The code editor pane in the center shows the contents of 'start.py'.

```
Day_1 > start.py
1 print ("Welcome to 21 day Python course!")
2
```

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL

```
● gitpod /workspace/python-for-OSINT-21-days (main) $ python Day_1/start.py
  Welcome to 21 day Python course!
○ gitpod /workspace/python-for-OSINT-21-days (main) $ 
```

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
```

The screenshot shows a code editor interface with the following details:

- EXPLORER:** Shows a file tree with a project named "PYTHON-FOR-OSINT-21-DAYS" containing subfolders "Day_1", "Day_2", "Day_3", and "Day_4". Inside "Day_2", there are files "start.py", "condition.py", "list.py", "loop.py", and "variable.py".
- EDITOR:** The file "variable.py" is open, displaying the following code:

```
Day_2 > variable.py
1 first_name = "John"
2 last_name = input('What is your last name?\n')
3 print("You are" + " " + first_name + " " + last_name)
4
```
- TERMINAL:** Shows the command-line session:

```
gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_2
gitpod /workspace/python-for-OSINT-21-days/Day_2 (main) $ python variable.py
What is your last name?
Smith
You are John Smith
gitpod /workspace/python-for-OSINT-21-days/Day_2 (main) $
```

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:

```
variable.py M condition.py X
Day_2 > condition.py
1 age = input('How old are you?\n')
2
3
4 if int(age) > 27:
5     print("You are so old")
6 elif int(age) < 27:
7     print("You are so young")
8
```

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL

```
gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_2
gitpod /workspace/python-for-OSINT-21-days/Day_2 (main) $ python condition.py
How old are you?
26
You are so young
gitpod /workspace/python-for-OSINT-21-days/Day_2 (main) $ python condition.py
How old are you?
29
You are so old
gitpod /workspace/python-for-OSINT-21-days/Day_2 (main) $
```

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`:

```
Day_2 > list.py
1 girls = ["Anna", "Maria", "Eva"]
2 print(girls)
3
4 girls.append("Brenda")
5 print(girls)
6
7 print (girls[3])
8
```

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL

- gitpod /workspace/python-for-OSINT-21-days (main) \$ cd Day_2
- gitpod /workspace/python-for-OSINT-21-days/Day_2 (main) \$ python list.py
['Anna', 'Maria', 'Eva']
['Anna', 'Maria', 'Eva', 'Brenda']
Brenda
- gitpod /workspace/python-for-OSINT-21-days/Day_2 (main) \$

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`:

The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left shows a file tree with a folder named 'Day_2' containing files like 'list.py', 'loop.py', 'start.py', 'condition.py', and 'list.py'. The 'OPEN EDITORS' section shows 'list.py Day_2' and 'loop.py Day_2'. The main editor area displays the content of 'loop.py':

```
Day_2 > loop.py
1  girls = ["Anna", "Maria", "Eva"]
2
3  for girl in girls:
4      print(girl +"; ")
5
6
7  for x in range(20):
8      print(x)
9
```

The terminal at the bottom shows the output of running the script:

```
gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_2
gitpod /workspace/python-for-OSINT-21-days/Day_2 (main) $ python loop.py
Anna;
Maria;
Eva;
0
1
2
3
4
5
6
7
8
9
10
11
12
```

```
# 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.

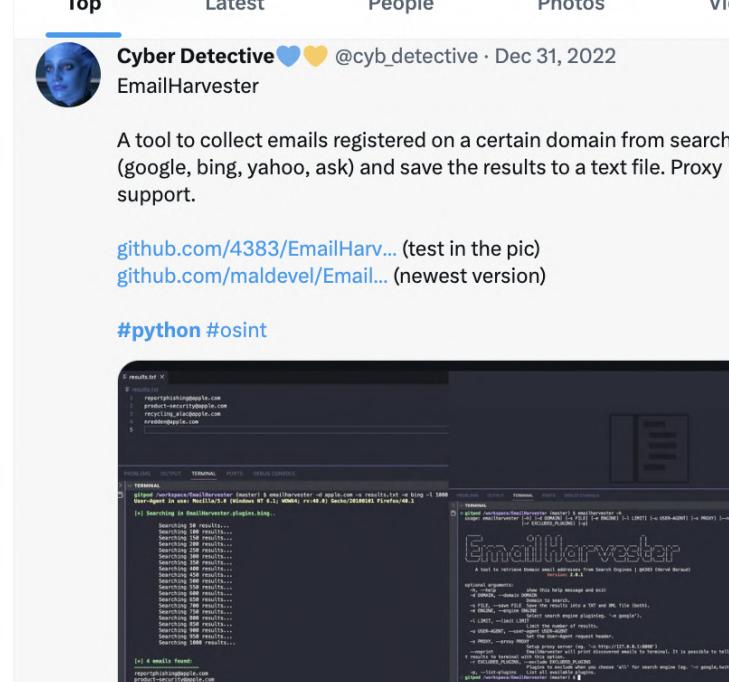
Day 3. Install and run Python command line tools

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.



- Home
- # Explore
- Notifications
- Messages
- Bookmarks
- Twitter Blue
- Verified Organizations
- Profile
- More

Tweet



← #python from:cyb_detective ...

Top Latest People Photos Videos

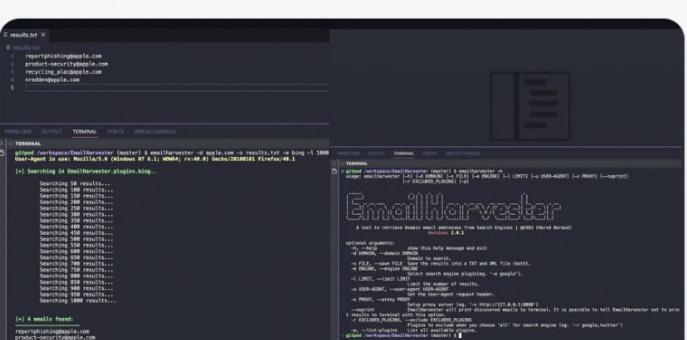
Cyber Detective   @cyb_detective · Dec 31, 2022

EmailHarvester

A tool to collect emails registered on a certain domain from search results (google, bing, yahoo, ask) and save the results to a text file. Proxy support.

github.com/4383/EmailHarv... (test in the pic)
github.com/maldevel/Email... (newest version)

#python #osint



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
```

```
● gitpod /workspace/python-for-OSINT-21-days (main) $ pip install thorndyke
Collecting thorndyke
  Downloading thorndyke-2022.1.2.1-py2.py3-none-any.whl (23 kB)
Requirement already satisfied: requests in /home/gitpod/.pyenv/versions/3.11.1/lib/python3.11/site-packages (from thorndyke) (2.28.2)
Collecting tqdm (from thorndyke)
  Downloading tqdm-4.65.0-py3-none-any.whl (77 kB)
    77.1/77.1 kB 3.9 MB/s eta 0:00:00
Requirement already satisfied: charset-normalizer<4,>=2 in /home/gitpod/.pyenv/versions/3.11.1/lib/python3.11/site-packages (from requests->thorndyke) (3.1.0)
Requirement already satisfied: idna<4,>=2.5 in /home/gitpod/.pyenv/versions/3.11.1/lib/python3.11/site-packages (from requests->thorndyke) (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->thorndyke) (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->thorndyke) (2022.12.7)
Installing collected packages: tqdm, thorndyke
Successfully installed thorndyke-2022.1.2.1 tqdm-4.65.0
```

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

THORNDYKE

[~] Enumerating @johmsmith on:
  0%|          | 0/233 [00:00<?, ?it/s]
[-] YouTube: Not Found
[-] Facebook: Not Found
  1%|■         | 1/233 [00:00<00:28,  8.18it/s]
[-] Instagram: Not Found
  1%|■         | 2/233 [00:00<01:24,  2.73it/s]
[-] TikTok: Not Found
[-] Likee: Not Found
  2%|■■        | 3/233 [00:00<01:06,  3.43it/s]
[-] Snapchat: Not Found
[+] Pinterest: https://www.pinterest.com/johmsmith/
  3%|■■■       | 4/233 [00:01<01:02,  3.59it/s]
[-] Wikipedia: Not Found
  3%|■■■       | 5/233 [00:01<01:06,  3.43it/s]
[-] Wattpad: Not Found
  4%|■■■■      | 6/233 [00:02<00:55,  4.02it/s]
[-] Google PlayStore: Not Found
  4%|■■■■      | 7/233 [00:02<00:51,  4.34it/s]

```

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.

```
● gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_3  
● gitpod /workspace/python-for-OSINT-21-days/Day_3 (main) $ cd blackbird  
○ gitpod /workspace/python-for-OSINT-21-days/Day_3/blackbird (main) $ python blackbird.py -u ivanov
```



Made with ❤ by p1ngul1n0

```
[!] Searching 'ivanov' across 582 social networks  
[+] - #4 Telegram account found - https://t.me/ivanov [200 OK]  
|--Name: Ivanov  
[+] - #11 Soundcloud account found - https://soundcloud.com/ivanov [200 OK]  
|--Name: Ivan Ivanov  
|--location: Yekaterinburg, Russian Federation  
|--picture:  
[+] - #38 Disqus account found - https://disqus.com/by/ivanov/ [200 OK]  
[+] - #10 Reddit account found - https://www.reddit.com/user/ivanov/about.json [200 OK]  
|--Name:  
|--Bio:  
|--picture:  
[+] - #23 Pastebin account found - https://pastebin.com/u/ivanov [200 OK]  
|--Member since: Thursday 4th of October 2012 07:53:53 AM CDT  
|--picture: https://pastebin.com/cache/img/4/7/20/340792.jpg  
[+] - #13 Steam account found - https://steamcommunity.com/id/ivanov/ [200 OK]  
|--Name:
```

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

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

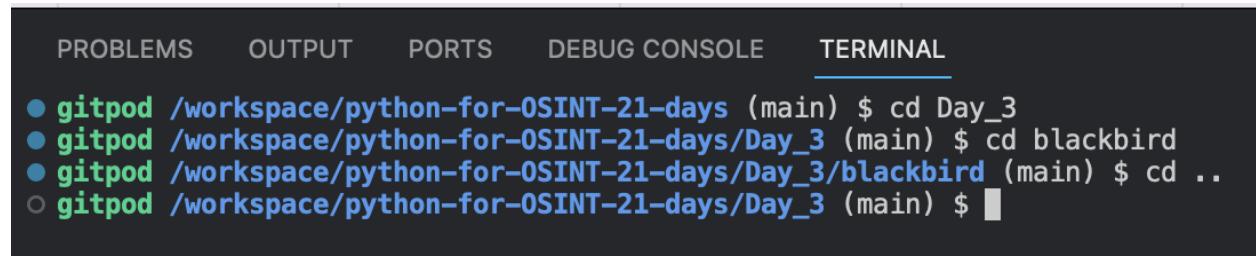
Third way

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.

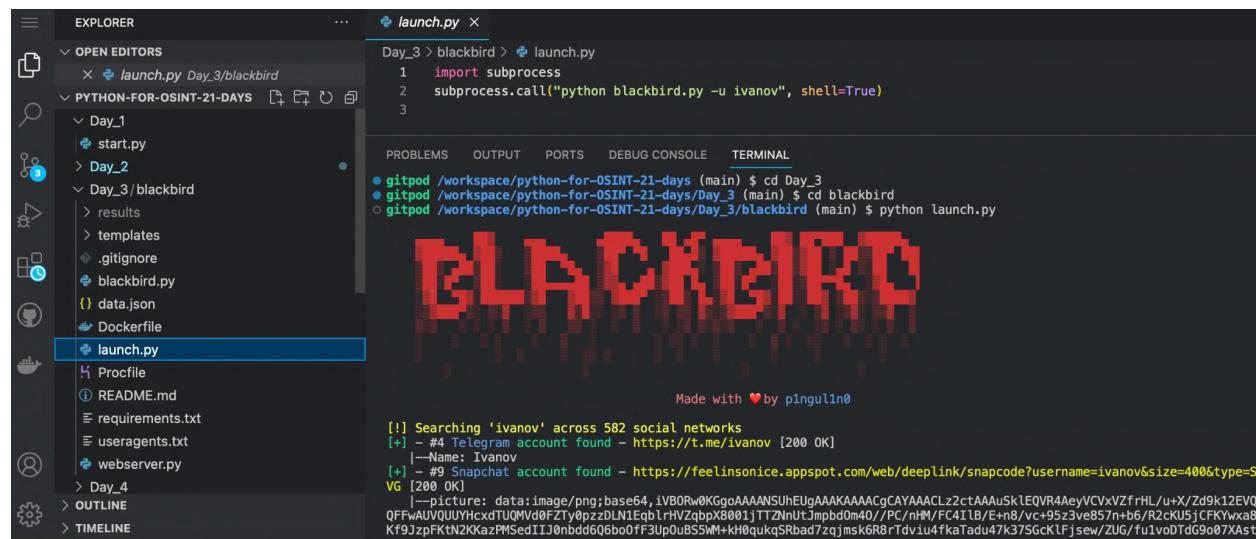


```
PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL
● gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_3
● gitpod /workspace/python-for-OSINT-21-days/Day_3 (main) $ cd blackbird
● gitpod /workspace/python-for-OSINT-21-days/Day_3/blackbird (main) $ cd ..
○ gitpod /workspace/python-for-OSINT-21-days/Day_3 (main) $ █
```

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

```
cd ..
```

Run **launch.py**:



```
EXPLORER          launch.py ×
OPEN EDITORS      Day_3 > blackbird > launch.py
PYTHON-FOR-OSINT-21-DAYS
Day_1
  start.py
Day_2
Day_3/blackbird
  > results
  > templates
  .gitignore
  blackbird.py
  {} data.json
  Dockerfile
  launch.py
  Procfile
  README.md
  requirements.txt
  useragents.txt
  webserver.py
Day_4
OUTLINE
TIMELINE
```

```
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) $ cd blackbird
gitpod /workspace/python-for-OSINT-21-days/Day_3/blackbird (main) $ python launch.py
```

Made with ❤ by p1ngulin0

```
[!] Searching 'ivanov' across 582 social networks
[+] - #4 Telegram account found - https://t.me/ivanov [200 OK]
|--Name: Ivanov
[+] - #3 Snapchat account found - https://feelinsonice.appspot.com/web/deeplink/snapcode?username=ivanov&size=400&type=SVG [200 OK]
|--picture: data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAAkAAAAACgCAYAAACLz2ctAAuSkLEQVR4AeyVCvXZfRLu+x/zd9k12EV0QFFWAUVQUYHcxdtUQWd0FZTy0pzDLN1Eqb1rhVZqpb8001jTTZnUJjmpbd0m40//PC/nHM/Fc41B/E+n8/vc+95z3ve857n+b6/R2cKU5jCFKWyxa8Kf9JzpFKtN2KKazPMSe1IIJ0nbddQ6bo0ff3Up0uBS5W+KH0qukqSRbad7zaJmsk6R8r7dviu4fkaTadu47k37SGckLFjsew/ZUG/fu1voTDg9o07XAst
```

```
# 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.

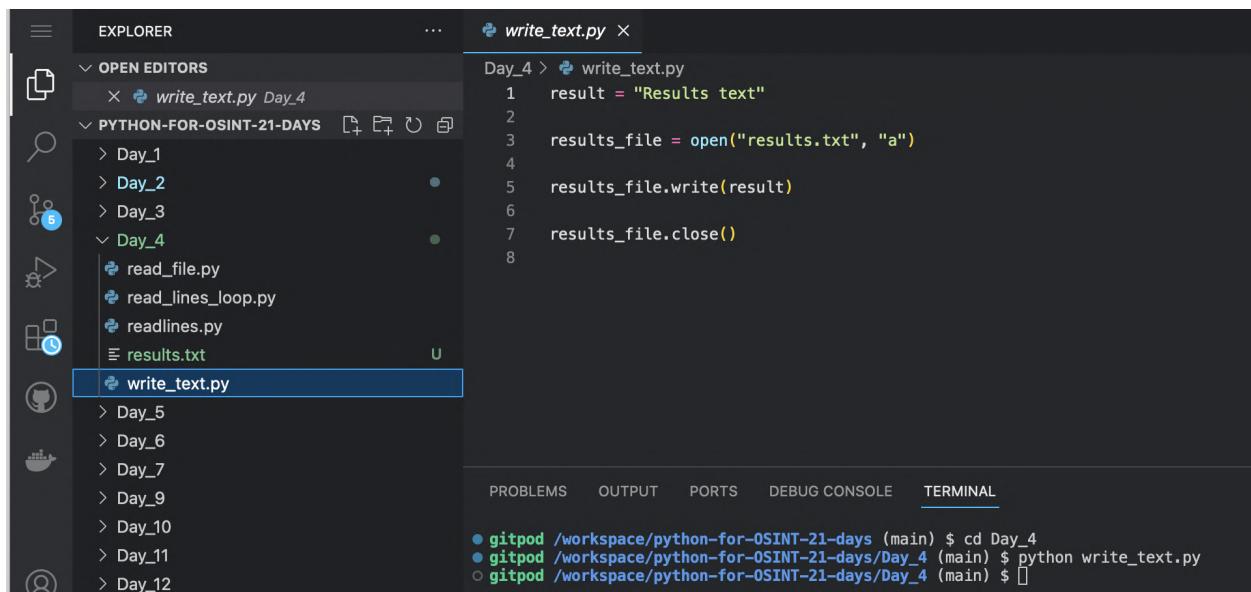
Day 4. Reading and writing files

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**:



```
Day_4 > write_text.py
1 result = "Results text"
2
3 results_file = open("results.txt", "a")
4
5 results_file.write(result)
6
7 results_file.close()
```

gitpod /workspace/python-for-OSINT-21-days (main) \$ cd Day_4
gitpod /workspace/python-for-OSINT-21-days/Day_4 (main) \$ python 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:

```

1
2  v [97m_
3  |  |  |  |  |  |  |  |  |  |  |  |  |  |
4  | 31;1m  |  |  |  |  |  |  |  |  |  |  |  |
5  | 0m
6  [97m[ 92m+ [97m] Enumerating @ 92mivanov [97m on: 0m
7  [97m[ 31;1m- [97m] YouTube: 31;1mNot Found 0m
8  [97m[ 92m+ [97m] Facebook: 92mhttps://www.facebook.com/ivanov 0m
9  [97m[ 31;1m- [97m] Instagram: 31;1mNot Found 0m
10 [97m[ 31;1m- [97m] TikTok: 31;1mNot Found 0m
11 [97m[ 31;1m- [97m] Likee: 31;1mNot Found 0m
12 [97m[ 31;1m- [97m] Snapchat: 31;1mNot Found 0m
13 [97m[ 92m+ [97m] Pinterest: 92mhttps://www.pinterest.com/ivanov 0m
14 [97m[ 31;1m- [97m] Wikipedia: 31;1mNot Found 0m
15 [97m[ 31;1m- [97m] Wattpad: 31;1mNot Found 0m
16 [97m[ 31;1m- [97m] Google PlayStore: 31;1mNot Found 0m

```

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL

```

● gitpod /workspace/python-for-OSINT-21-days (main) $ thorndyke ivanov >ivanov_pages.txt
100% | 233/233 [05:56<00:00, 1.53s/it]
○ gitpod /workspace/python-for-OSINT-21-days (main) $ []

```

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

Reading file

Run `read_file.py`:

```

Day_4 > ⚡ read_file.py
1   results_file = open("results.txt", "r")
2
3   print(results_file.read())
4

```

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL

```

● gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_4
● gitpod /workspace/python-for-OSINT-21-days/Day_4 (main) $ python read_file.py
Results text
○ gitpod /workspace/python-for-OSINT-21-days/Day_4 (main) $ 

```

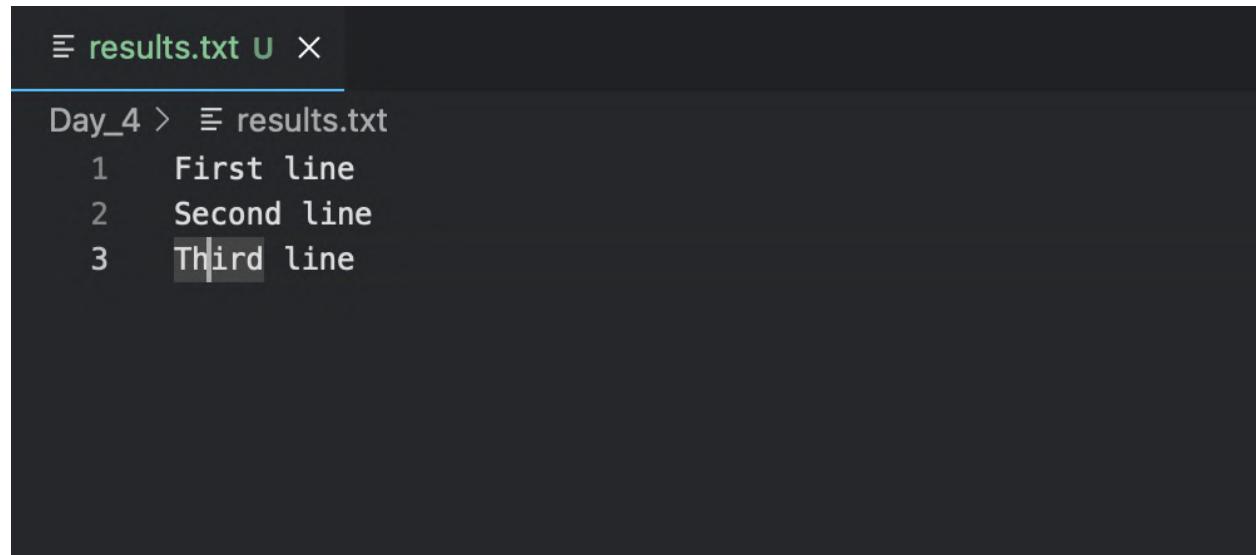
Open the `results.txt` file:

```
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.



The screenshot shows a terminal window with the title bar 'results.txt'. The window displays the following text:

```
Day_4 > results.txt
1 First line
2 Second line
3 Third line
```

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

```
1 stringNumber = 1
2
3
4 with open("results.txt") as f:
5     for line in f:
6         print(str(stringNumber) + ". " + line)
7         stringNumber += 1
```

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)
```

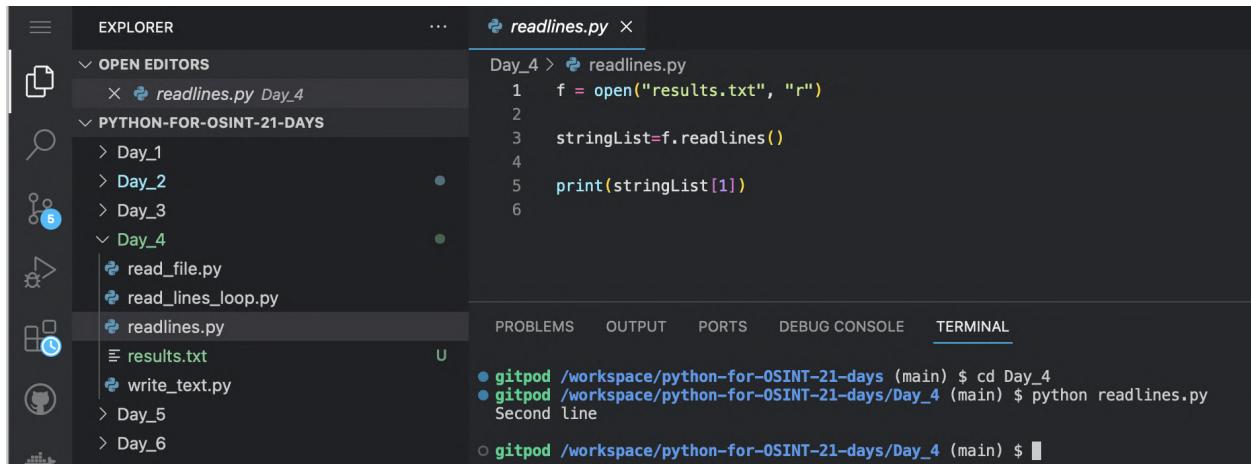
Increase the line number by one:

```
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`:



The screenshot shows the VS Code interface. The Explorer sidebar on the left lists files and folders, including `readlines.py` under Day_4. The code editor on the right contains the following Python code:

```
Day_4 > readlines.py
1 f = open("results.txt", "r")
2
3 stringList=f.readlines()
4
5 print(stringList[1])
6
```

The terminal at the bottom shows the command being run and its output:

```
gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_4
gitpod /workspace/python-for-OSINT-21-days/Day_4 (main) $ python readlines.py
Second line
gitpod /workspace/python-for-OSINT-21-days/Day_4 (main) $
```

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 screenshot shows a web application interface for making HTTP requests. At the top, there are navigation links: REST test test..., Source Code, Submit Bug, and Author. Below this is a form titled "HTTP request options". It has two input fields: "Method" set to "GET" and "Endpoint" set to "https://api.github.com/search/users?q=javascript". A note below says "Method and Endpoint are required. Click below to add additional parameters." There are three buttons: "Add authentication", "Add header", and "Add parameter". A "File" button is also present. A large green "Ajax request" button is at the bottom right of the form. To the right of the form is a "HTTP 200 success" status indicator. Below it is a JSON response from GitHub's search API. At the bottom right is a detailed view of the response headers.

```
{  
  "total_count": 128698,  
  "incomplete_results": false,  
  "items": [  
    {  
      "login": "javascript",  
      "id": 48139711,  
      "node_id": "MDEyOk9yZzFuaXphdGlvbjQ4MTMSNzEx",  
      "avatar_url": "https://avatars.githubusercontent.com/u/48139711?v=4",  
      "gravatar_id": "",  
      "url": "https://api.github.com/users/javascript",  
      "html_url": "https://github.com/javascript",  
      "followers_url": "https://api.github.com/users/javascript/followers",  
      "following_url": "https://api.github.com/users/javascript/following{/other_user}",  
      "gists_url": "https://api.github.com/users/javascript/gists{/gist_id}"  
    },  
    ...  
  ],  
  "starred_url": "https://api.github.com/users/javascript/starred{/owner}"  
}
```

```
cache-control: no-cache  
content-type: application/json; charset=utf-8  
link: <https://api.github.com/search/users?q=javascript&page=2>; rel="next",  
<https://api.github.com/search/users?q=javascript&page=34>; rel="last"  
x-github-media-type: github.v3; format=json  
x-github-request-id: 5C4B:BF6F:7C0EBC6:7D39AAD:644BDF00  
x-ratelimit-limit: 10  
x-ratelimit-remaining: 9  
x-ratelimit-reset: 1682693948  
x-ratelimit-resource: search  
x-ratelimit-used: 1
```

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.

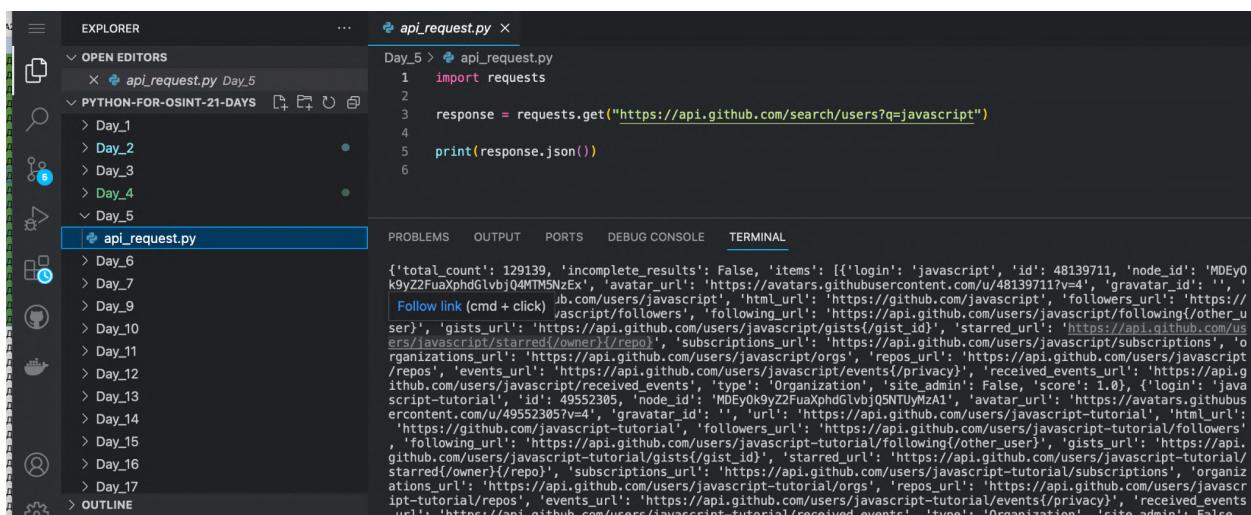
For this we will use packages requests (<https://pypi.org/project/requests/>).

```
● 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-packages (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 requests) (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 (from 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
 - PastebinLeaks
 - 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/Surname | | | |
|------------------|---|---|--------------------------------|----------------|---|---|---------------------|
| Name | Link | Description | Price | Name | Link | Description | Price |
| ipstack | https://ipstack.com | Detect country, region, city and zip code | FREE | 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 |
| Ipgeolocation.io | https://ipgeolocation.io | Provides country, city, state, province, location, currency, latitude and longitude, company detail, ISP lookup, language, zip code, time zone, current time, sunset and sunrise time, moonrise and moonset | 30 000 requests per month/FREE | Agify.io | https://agify.io | Predicts the age of a person given their name | 1000 names/day free |
| IPInfoDB | https://ipinfodb.com/api/ | Free Geolocation tools and APIs for country, region, city and time zone lookup by IP address | FREE | Nationalize.io | https://nationalize.io | Predicts the nationality of a person given their name | 1000 names/day free |
| IP API | https://ip-api.com/ | Free domain/IP geolocation info | FREE | | | | |

| Wi-fi lookup | | | | PastebinLeaks | | | |
|--------------|---|--|-------|---------------------|---|--|-----------------------------------|
| Name | Link | Description | Price | Name | Link | Description | Price |
| Mylinkov API | https://www.mylinkov.org | public API implementation of Wi-Fi Geo-Location database | FREE | HaveIBeenPwned | https://haveibeenpwned.com/APIv3 | allows the list of pwned (email addresses and usernames) | \$3.50 per month |
| Wigle | https://api.wigle.net/ | get location and other information by SSID | FREE | Psdmp.ws | https://psdmp.ws/api | search in Pastebin | \$0.95 per 10000 requests |
| | | | | LeakPeek | https://psdmp.ws/api | search in leaks databases | \$0.99 per weeks unlimited access |
| | | | | BreachDirectory.com | https://breachdirectory.com/api/?lang=en | search_domain in data breaches databases | FREE |
| | | | | | | search_domain, email_address, full_name, ip_address, phone, password, user_name in leaks databases | 10 requests FREE |
| | | | | LeakLookup | https://leak-lookup.com/api | | |
| | | | | BreachDirectory.org | https://rapidapi.com/rohan-patra/api/breacheddirectory/pricing | search_domain, email_address, full_name, ip_address, password, user_name in leaks databases | 50 requests in monthly/FREE |

| Network | | | | Finance | | | |
|-------------|---|---|-------|--------------------|---|--|-------|
| Name | Link | Description | Price | Name | Link | Description | Price |
| PeeringDB | https://www.peeringdb.com/apis/docs/ | Database of networks, and the go-to location for interconnection points | FREE | Binlist.net | https://binlist.net/ | get information about bank by BIN | FREE |
| PacketTotal | https://packettotal.com/api.html | :pcap files analyze | 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 |

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.

REQBIN

EXAMPLES SAVED

POST Request Example
Bearer Token Auth Example
HTML Form POST Example
GET Request Example
REST API POST Example
PUT Request Example
Auth Bearer Header Example
POST API Example
POST JSON Example
JSON Payload Example
Content-Length Example
REST API GET Example
Send Cookies Example
POST XML Example
JSON Response Example
POST JSON with Bearer Token
Curl POST Request Example
Curl GET Request Example
Curl POST JSON Example
Curl POST Body Example
Curl Basic Auth Example
Curl Bearer Token Example
Curl Download File Example
Curl Ignore SSL Checks

Curl Python JavaScript Node.js PHP Java JSON XML

Online REST & SOAP API Testing Tool

ReqBin is an online API testing tool for REST and SOAP APIs. Test API endpoints by making API requests directly from your browser. Test API responses with built-in JSON and XML validators. Load test your API with hundreds of simulated concurrent connections. Generate code snippets for API automation testing frameworks. Share and discuss your API requests online.

File /> Generate Code Tools Share /> Generate Code App Mode

https://api.github.com GET US Send

Authorization Content-Type Raw (2)

Bearer Token Basic Auth No Auth

Token:

The authorization header will be automatically generated when you send the request. Read more about HTTP Authentication.

Status: 200 (OK) Time: 129 ms Size: 34.50 kb

Content (635) Headers (26) Raw (663) JSON

Timings

```
total_count": 128399,
"incomplete_results": false,
"items": [
  {
    "login": "javascript",
    "id": 48139711,
    "node_id": "MDEy0k9yZFuaxphdGlvbjQ4MTM5NzEx",
    "avatar_url": "https://avatars.githubusercontent.com/u/48139711?v=4",
    "gravatar_id": "",
    "url": "https://api.github.com/users/javascript",
    "html_url": "https://github.com/javascript",
    "followers_url": "https://api.github.com/users/javascript/followers",
    "following_url": "https://api.github.com/users/javascript/following"
  }
]
```

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](#)).

The screenshot shows a JSON viewer interface with two panes. The left pane displays the raw JSON code, and the right pane shows the hierarchical structure of the JSON object.

Raw JSON (Left Pane):

```
1- [
2   "total_count": 128992,
3   "incomplete_results": false,
4   "items": [
5     {
6       "login": "javascript",
7       "id": 48139711,
8       "node_id": "MDEyOk9yZ2FuaXphdGlvbjQ4MTM5NzEx",
9       "avatar_url": "https://avatars.githubusercontent.com/u/48139711?v=4",
10      "gravatar_id": "",
11      "url": "https://api.github.com/users/javascript",
12      "html_url": "https://github.com/javascript",
13      "followers_url": "https://api.github.com/users/javascript/followers",
14      "following_url": "https://api.github.com/users/javascript/following/other_user",
15      "gists_url": "https://api.github.com/users/javascript/gists{/gist_id}",
16      "starred_url": "https://api.github.com/users/javascript/starred{/owner}{/repo}",
17      "subscriptions_url": "https://api.github.com/users/javascript/subscriptions",
18      "organizations_url": "https://api.github.com/users/javascript/orgs",
19      "repos_url": "https://api.github.com/users/javascript/repos"
]
```

JSON Viewer (Right Pane):

```
object > items >
object {3}
  total_count : 128992
  incomplete_results : false
  items [30]
    object {19}
      login : javascript
      id : 48139711
      node_id : MDEyOk9yZ2FuaXphdGlvbjQ4MTM5NzEx
      avatar_url : https://avatars.githubusercontent.com/u/48139711?v=4
      gravatar_id : value
      url : https://api.github.com/users/javascript
      html_url : https://github.com/javascript
      followers_url : https://api.github.com/users/javascript/followers
      following_url : https://api.github.com/users/javascript/following/other_user
      gists_url : https://api.github.com/users/javascript/gists{/gist_id}
      starred_url : https://api.github.com/users/javascript/starred{/owner}{/repo}
      subscriptions_url : https://api.github.com/users/javascript/subscriptions
      organizations_url : https://api.github.com/users/javascript/orgs
      repos_url : https://api.github.com/users/javascript/repos
      events_url : https://api.github.com/users/javascript/events
      received_events_url : https://api.github.com/users/javascript/received_events
    }
  type : Organization
```

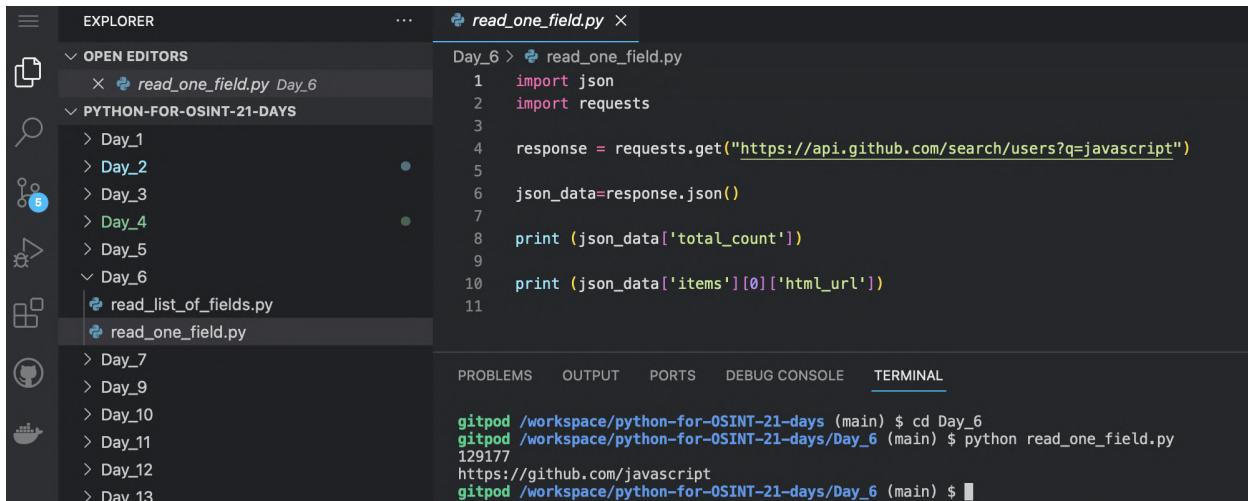
In response to the query, we get a list of 30 objects (items[0], items[1], items[2] etc), each corresponding to a specific user.

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`:



The screenshot shows a code editor interface with the following details:

- EXPLORER** sidebar:
 - OPEN EDITORS**: `read_one_field.py Day_6`
 - PYTHON-FOR-OSINT-21-DAYS** folder:
 - > Day_1
 - > Day_2
 - > Day_3
 - > Day_4
 - > Day_5
 - > Day_6
 - > Day_7
 - > Day_9
 - > Day_10
 - > Day_11
 - > Day_12
 - > Day_13
 - `read_list_of_fields.py`
 - `read_one_field.py`
- Code Editor**:

```
Day_6 > ⚡ read_one_field.py
1 import json
2 import requests
3
4 response = requests.get("https://api.github.com/search/users?q=javascript")
5
6 json_data=response.json()
7
8 print (json_data['total_count'])
9
10 print (json_data['items'][0]['html_url'])
11
```
- Tabs**: PROBLEMS, OUTPUT, PORTS, DEBUG CONSOLE, TERMINAL
- Terminal**:

```
gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_6
gitpod /workspace/python-for-OSINT-21-days/Day_6 (main) $ python read_one_field.py
129177
https://github.com/javascript
gitpod /workspace/python-for-OSINT-21-days/Day_6 (main) $
```

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'])
```

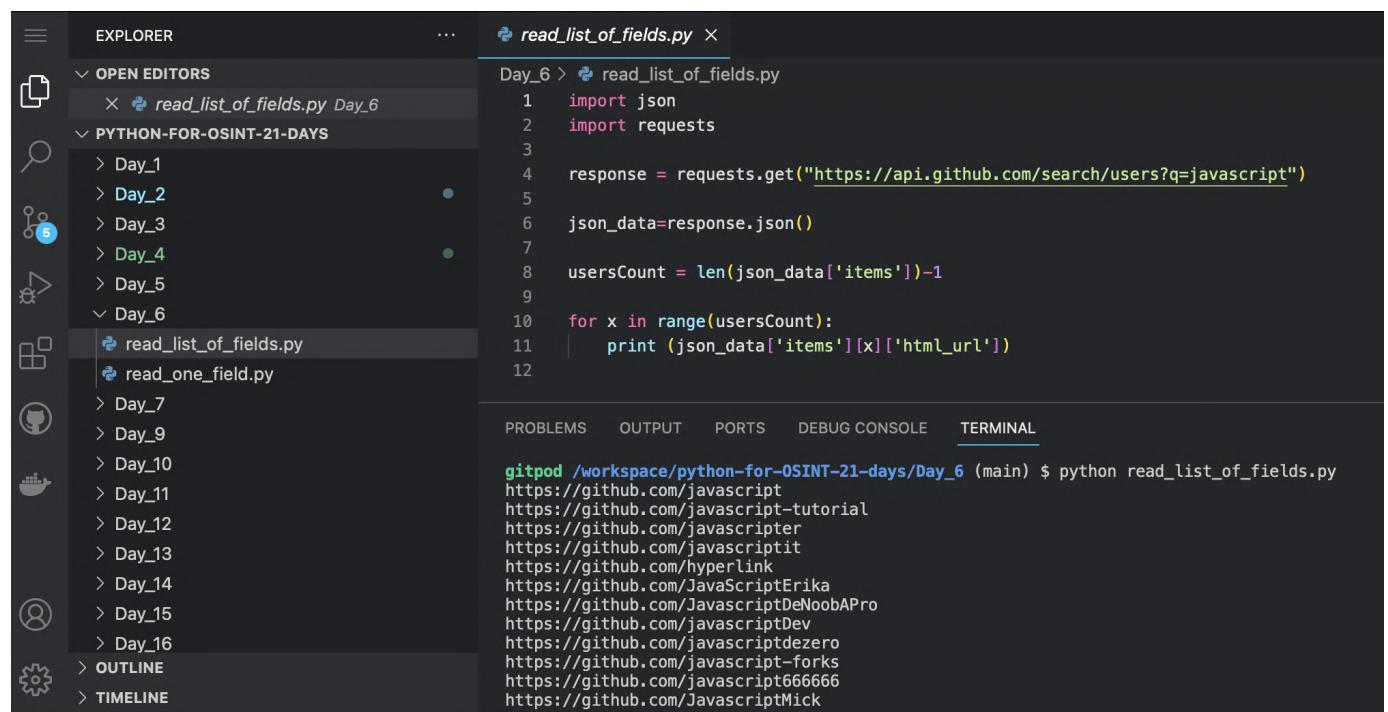
Output the link to the first Github profile from the results:

```
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`:



The screenshot shows a code editor interface with the following details:

- EXPLORER:** Shows a file tree with a folder named "PYTHON-FOR-OSINT-21-DAYS" containing subfolders "Day_1" through "Day_6". Inside "Day_6", there are files "read_list_of_fields.py" and "read_one_field.py".
- Code Editor:** The file "read_list_of_fields.py" is open, displaying Python code to search for "javascript" users on GitHub and print their profiles. The code uses the `requests` library to make a GET request to the GitHub API and `json` to parse the response.
- Terminal:** Below the code editor, the terminal window shows the command being run: `gitpod /workspace/python-for-OSINT-21-days/Day_6 (main) $ python read_list_of_fields.py`. The output lists 15 GitHub user profiles starting with "javascript".

```
gitpod /workspace/python-for-OSINT-21-days/Day_6 (main) $ python read_list_of_fields.py
https://github.com/javascript
https://github.com/javascript-tutorial
https://github.com/javascripter
https://github.com/javascriptit
https://github.com/hyperlink
https://github.com/JavaScriptErika
https://github.com/JavascriptDeNoobAPro
https://github.com/javascriptDev
https://github.com/javascriptdezero
https://github.com/javascript-forks
https://github.com/javascript666666
https://github.com/JavascriptMick
```

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'])
```

JSONPath Online Evaluator - jsonpath.com

JSONPath
\$.items[0]["html_url"]

Output paths [Expand JSONPath expressions](#)

Inputs

```
1 - {  
2   "total_count": 125524,  
3   "incomplete_results": false,  
4   "items": [  
5     {  
6       "login": "javascript",  
7       "id": 48139711,  
8       "node_id": "MDIyOkYzZ2PuaXphdGLvbjQ4MTM5NzEx",  
9       "avatar_url": "https://avatars.githubusercontent.com/u/48139711?v=4",  
10      "gravatar_id": "",  
11      "url": "https://api.github.com/users/javascript",  
12      "html_url": "https://github.com/javascript",  
13      "followers_url": "https://api.github.com/users/javascript/followers",  
14      "following_url": "https://api.github.com/users/javascript/following{/other_us  
15      "gists_url": "https://api.github.com/users/javascript/gists{/gist_id}"}
```

Evaluation Results

```
1 - []  
2 - "https://github.com/javascript"  
3 - []
```

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.

```
1 |created_on,updated_from_linkedin,full_name,first_name,last_name,headline,current_position,company_name ,person_city,person_state,person_country,person_industry,tags,person_skills,education_experience,compa ny_website,email1,email1_type,email1_status,email2,email2_type,email2_status,email3,email3_type,email3 _status,email4,email4_type,email4_status,phone1,phone1_type,phone1_status,phone2,phone2_type,phone2_st atus,phone3,phone3_type,phone3_status,phone4,phone4_type,phone4_status,company_size_from,company_size_ to,current_position_2,current_company_2,previous_position_2,previous_company_2,previous_position_3,pre vious_company_3,company_city,company_state,company_country,person_linkedin_url,person_angellist_url,pe rson_crunchbase_url,person_twitter_url,person_faceb ook_url,company_linkedin_url,person_image_url,company_logo_url,created_by,lead_status  
2 |2022-12-19,2022-12-19,Caitlin Maier,Caitlin,Maier,Assistant Vice President Talent Acquisition at Golub Capital,Assistant Vice President Talent Acquisition,Golub Capital,Chicago,Illinois,United States,Financial Services,"Recruiting, Customer Service, Sales, Management, Microsoft Word, Cold Calling, Entrepreneurship, Leadership, Microsoft Office, Microsoft Excel, Human Resources, Sales Process, Interviewing, Facebook, Training, Account Management, Marketing, Social Media, PowerPoint, Social Media Marketing, Onboarding",James Madison University (2007 - 2011),https://golubcapital.com/,maicerce@gmail.com,personal,valid,cmaier@golubcapital.com,main job,valid,,,,,,,,,,501,1000,,,Recruiting Coordinator,Capital One,Recruiter,Capital One,San Francisco,California,United States,https://www.linkedin.com/in/caitlinemaier,,,https://twitter.com/CaitlyMai_COF,,https://www.linkedin.com/company/
```

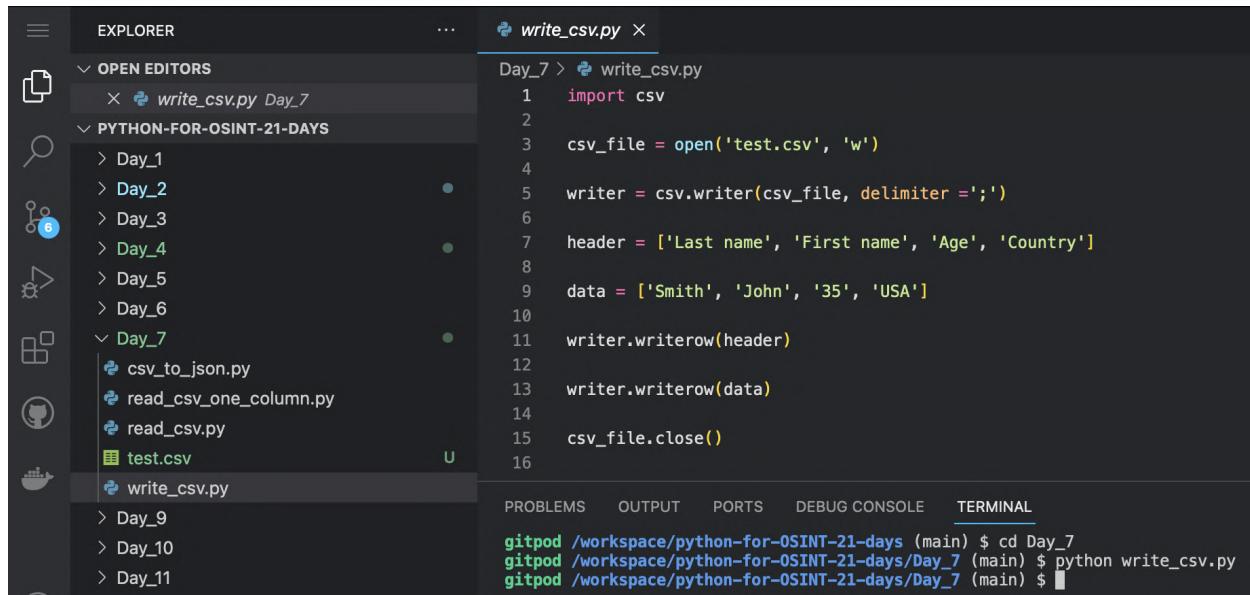
And this is how it looks when you open Numbers (MS Excel equivalent for Mac).

| 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 | Table data was imported. |
| 2022-12-19 | 2022-12-19 | Brian Boje | Brian | Boje | Vice President Of Recruiting at STS Technical Services | Adjust Settings |
| 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 Attractic | |

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:



The screenshot shows the VS Code interface. The Explorer sidebar on the left lists files and folders: Day_1, Day_2, Day_3, Day_4, Day_5, Day_6, Day_7, csv_to_json.py, read_csv_one_column.py, read_csv.py, test.csv, and write_csv.py. The Day_7 folder is expanded. The write_csv.py file is selected in the center editor area. The code in write_csv.py is:

```
import csv
csv_file = open('test.csv', 'w')
writer = csv.writer(csv_file, delimiter=',')
header = ['Last name', 'First name', 'Age', 'Country']
data = ['Smith', 'John', '35', 'USA']
writer.writerow(header)
writer.writerow(data)
csv_file.close()
```

The Terminal tab at the bottom shows the command being run: gitpod /workspace/python-for-OSINT-21-days (main) \$ cd Day_7, followed by gitpod /workspace/python-for-OSINT-21-days/Day_7 (main) \$ python 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)
```

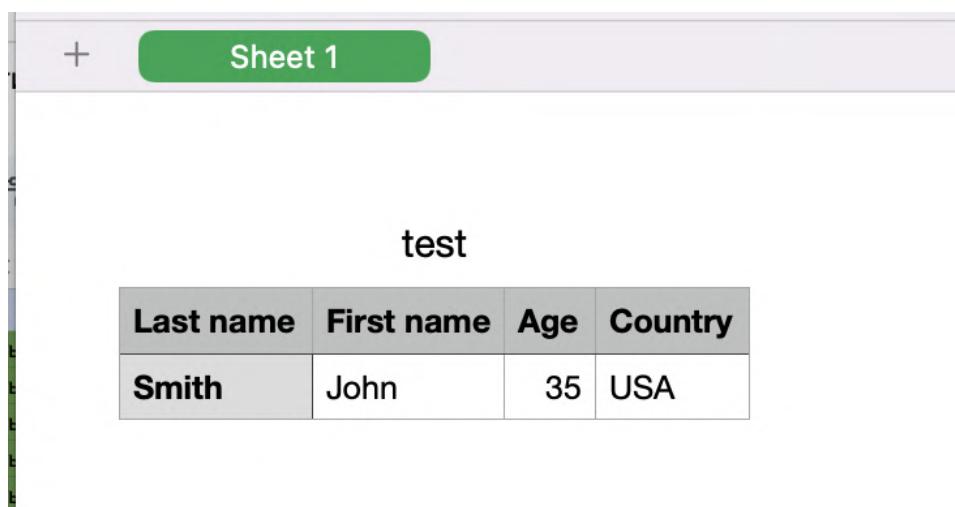
```
# Write a line of data to the file:
```

```
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.

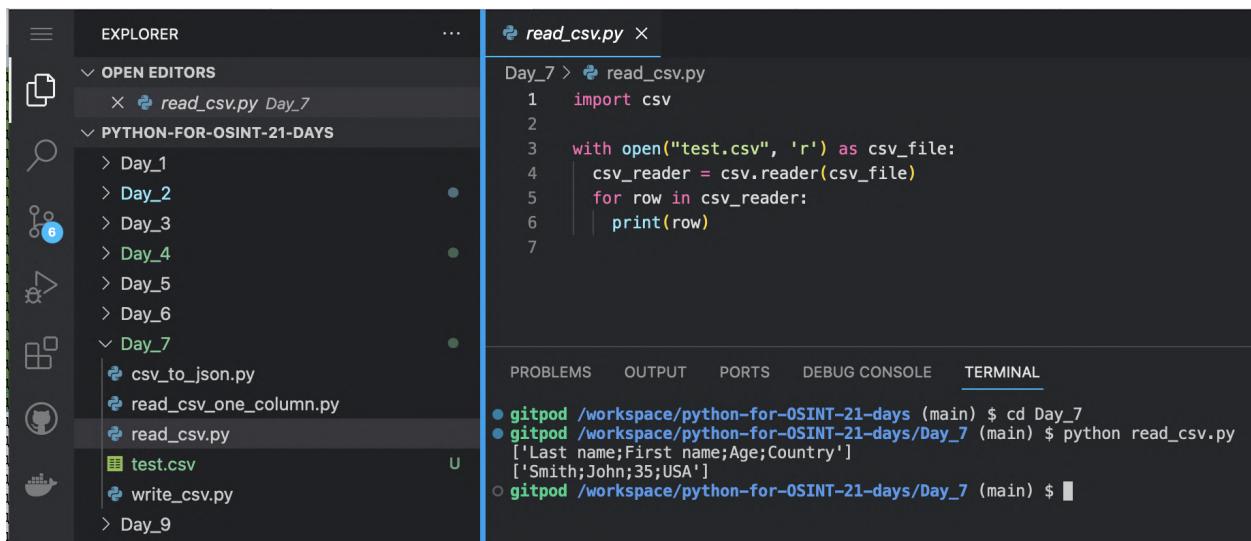


| Last name | First name | Age | Country |
|-----------|------------|-----|---------|
| Smith | John | 35 | USA |

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

Reading CSV file

Run `read_csv.py`:



```
Day_7 > read_csv.py
1 import csv
2
3 with open("test.csv", 'r') as csv_file:
4     csv_reader = csv.reader(csv_file)
5     for row in csv_reader:
6         print(row)
7
```

gitpod /workspace/python-for-OSINT-21-days (main) \$ cd Day_7
gitpod /workspace/python-for-OSINT-21-days/Day_7 (main) \$ python read_csv.py
['Last name;First name;Age;Country']
['Smith;John;35;USA']
gitpod /workspace/python-for-OSINT-21-days/Day_7 (main) \$

Import the csv package:

```
import csv
```

Open file test.csv:

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

Create csv.reader object:

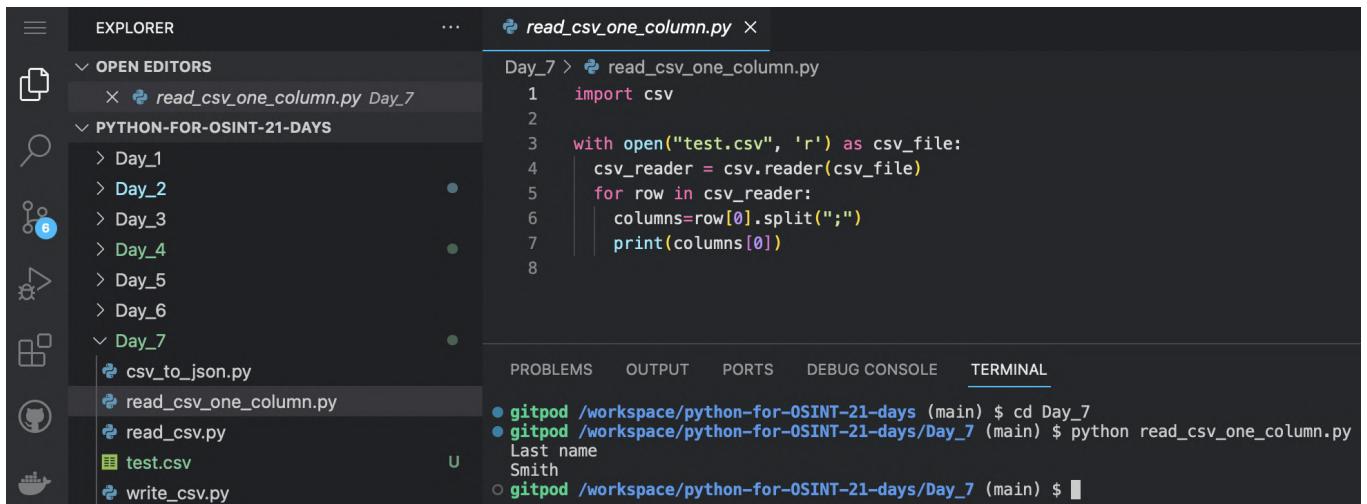
```
csv_reader = csv.reader(csv_file)
```

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`:



The screenshot shows the VS Code interface. On the left is the Explorer sidebar with a tree view of files and folders. In the center is the code editor showing a Python script named `read_csv_one_column.py`. The script reads a CSV file named `test.csv` and prints the first column of each row. On the right is the Terminal panel displaying the command `python read_csv_one_column.py` being run and its output, which shows the last name "Smith".

```
Day_7 > read_csv_one_column.py  
1 import csv  
2  
3 with open("test.csv", 'r') as csv_file:  
4     csv_reader = csv.reader(csv_file)  
5     for row in csv_reader:  
6         columns=row[0].split(";")  
7         print(columns[0])  
8  
PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL  
● gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_7  
● gitpod /workspace/python-for-OSINT-21-days/Day_7 (main) $ python read_csv_one_column.py  
Last name  
Smith  
○ gitpod /workspace/python-for-OSINT-21-days/Day_7 (main) $
```

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.

```
import json
import requests
import csv

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

csv_file = open('test.csv', 'w')
writer = csv.writer(csv_file, delimiter =';')

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

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)

csv_file.close()
```

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL

```
gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_7
gitpod /workspace/python-for-OSINT-21-days/Day_7 (main) $ python json_to_csv.py
gitpod /workspace/python-for-OSINT-21-days/Day_7 (main) $
```

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:
```

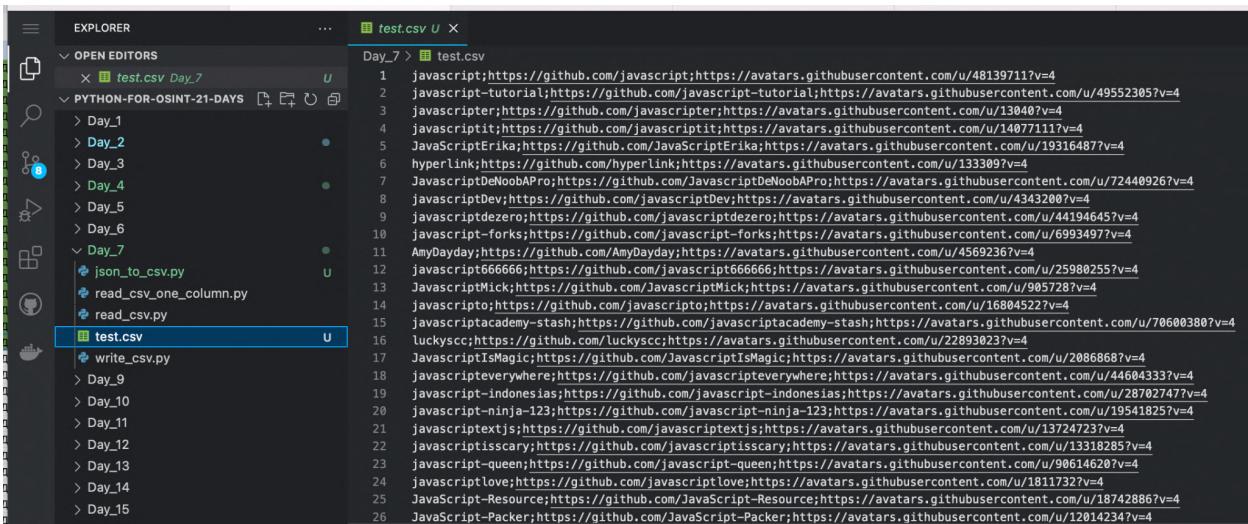
```
usersCount = len(json_data['items'])-1
```

```
# 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()
```



The screenshot shows the VS Code interface with the Explorer sidebar on the left and the Editor pane on the right. The Explorer sidebar lists several files and folders under 'OPEN EDITORS'. In the Editor pane, the file 'test.csv' is open, displaying a list of 26 entries, each consisting of a URL and a version number (v=4). The URLs are all variations of 'https://github.com/[repository]/blob/main/[file]'.

```
1 javascript;https://github.com/javascript;https://avatars.githubusercontent.com/u/48139711?v=4
2 javascript-tutorial;https://github.com/javascript-tutorial;https://avatars.githubusercontent.com/u/49552305?v=4
3 javascripter;https://github.com/javascripter;https://avatars.githubusercontent.com/u/13040?v=4
4 javascriptit;https://github.com/javascriptit;https://avatars.githubusercontent.com/u/14077117?v=4
5 JavaScriptErika;https://github.com/JavaScriptErika;https://avatars.githubusercontent.com/u/19316487?v=4
6 hyperlink;https://github.com/hyperlink;https://avatars.githubusercontent.com/u/1333097?v=4
7 JavascriptDeNoobAPro;https://github.com/JavascriptDeNoobAPro;https://avatars.githubusercontent.com/u/72440926?v=4
8 javascriptDev;https://github.com/javascriptDev;https://avatars.githubusercontent.com/u/43432007?v=4
9 javascriptdezero;https://github.com/javascriptdezero;https://avatars.githubusercontent.com/u/44194645?v=4
10 javascript-forks;https://github.com/javascript-forks;https://avatars.githubusercontent.com/u/6993497?v=4
11 AmyDayday;https://github.com/AmyDayday;https://avatars.githubusercontent.com/u/45692367?v=4
12 javascript66666;https://github.com/javascript66666;https://avatars.githubusercontent.com/u/25980255?v=4
13 JavascriptMick;https://github.com/JavascriptMick;https://avatars.githubusercontent.com/u/9057287?v=4
14 javascript;https://github.com/javascripto;https://avatars.githubusercontent.com/u/16804522?v=4
15 javascriptacademy-stash;https://github.com/javascriptacademy-stash;https://avatars.githubusercontent.com/u/70600380?v=4
16 luckyssc;https://github.com/luckyssc;https://avatars.githubusercontent.com/u/228930237?v=4
17 JavascriptIsMagic;https://github.com/JavascriptIsMagic;https://avatars.githubusercontent.com/u/20868687?v=4
18 javascripteverywhere;https://github.com/javascripteverywhere;https://avatars.githubusercontent.com/u/44604333?v=4
19 javascript-indonesias;https://github.com/javascript-indonesias;https://avatars.githubusercontent.com/u/28702747?v=4
20 javascript-ninja-123;https://github.com/javascript-ninja-123;https://avatars.githubusercontent.com/u/19541825?v=4
21 javascriptextjs;https://github.com/javascriptextjs;https://avatars.githubusercontent.com/u/13724723?v=4
22 javascriptisscary;https://github.com/javascriptisscary;https://avatars.githubusercontent.com/u/13318285?v=4
23 javascript-queen;https://github.com/javascript-queen;https://avatars.githubusercontent.com/u/98614620?v=4
24 javascriptLove;https://github.com/javascriptLove;https://avatars.githubusercontent.com/u/18117327?v=4
25 JavaScript-Resource;https://github.com/JavaScript-Resource;https://avatars.githubusercontent.com/u/18742886?v=4
26 JavaScript-Packer;https://github.com/JavaScript-Packer;https://avatars.githubusercontent.com/u/12014234?v=4
```

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.

The screenshot shows a Google search results page with the query "@gmail.com filetype:sql". The results list several links to SQL dump files:

- [ragebooter.txt](#)
... Dumping data for table `users` -- INSERT INTO `users` ('ID', 'username', ... 'anna-ellington-jones@hotmail.com', 0, 29, 1382677259, 0, "", 0), (108, ...)
- [GitLab](#)
https://gitlab.com › GitLab.org › GitLab ...
[db/structure.sql · master · GitLab.org / GitLab](#)
Quickly and easily edit multiple files in your project. ... Edit this file only.
- [alohachicas.com](#)
https://alohachicas.com › search · Translate this page ...
["也门高端男士休闲会所\(微信696360\) ...](#)
ALOHA SHOP. C/Velázquez Moreno 26. 36202 – Vigo (Pontevedra). +34 637 914 746. +34 692 219 176. pedidosalohashop@gmail.com ...
- [The University of Western Australia](#)
https://teaching.csse.uwa.edu.au › projects › juicdump3 ...
[SQL dump](#)

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.

| Google Hacking Database | | | |
|-------------------------|--|-----------------------------|-------------------------|
| | | Filters | |
| Show | 15 | Quick Search | sq |
| Date Added | Dork | Category | Author |
| 2023-04-20 | Re: Thank you for your submission! Re: intitle:index of db.sqlite3 | Files Containing Juicy Info | Shebu |
| 2023-03-14 | intitle:"index of" "database.sql" | Files Containing Juicy Info | Prathamesh Pawar |
| 2023-02-22 | inurl:backup filetype:sql | Files Containing Juicy Info | Nox Mentor |
| 2023-02-15 | intext:"index of" "backupop/*.sql" | Files Containing Juicy Info | Ahmad Kataranje |
| 2023-02-01 | inurl: administrator/components/com_admin/sql/updates/sqlazure | Files Containing Juicy Info | Mark Ivan David |
| 2023-02-01 | inurl: administrator/components/com_admin/sql/updates/mysql/ | Files Containing Juicy Info | Mark Ivan David |
| 2022-09-19 | intext:"index of" ".sql" | Files Containing Juicy Info | Gopalsamy Rajendran |
| 2022-06-16 | "index of" filetype:sql | Files Containing Juicy Info | Girish B O |
| 2022-06-16 | inttitle:"index of" filetype:sql | Files Containing Juicy Info | Ract Hack |
| 2022-06-16 | site:com.* inttitle:"index of" *.sql | Files Containing Juicy Info | Girish B O |
| 2022-06-16 | inttext:"SQL" && "DB" inurl:"/runtime/log/" | Files Containing Juicy Info | Vitor guaxi |
| 2021-11-15 | inttitle:"index of" "/mysql" | Files Containing Juicy Info | Priyanshu Choudhary |
| 2021-11-10 | inttitle:"index of" " mod_auth_mysql " | Files Containing Juicy Info | Muhammad Al-Amin |
| 2021-11-08 | inttitle:"database" "backup" filetype:sql | Files Containing Juicy Info | Onkar Deshmukh |
| 2021-11-05 | inurl:admin ext:sql | Files Containing Juicy Info | Veeresh Appasaheb Patil |

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](#).

 RebaseData

Converters API Libraries Pricing Security Support About [Login](#) [Register](#)

Convert MySQL to CSV online

Choose the .SQL file..

[Convert](#)

Input file

Our API uses a .SQL file as input. This could be a MySQL dump from the mysqldump tool or an output of any other tool that returns SQL. The file should contain CREATE TABLE and INSERT/UPDATE statements.

Max file size for web uploads: 50 GB
[Register](#) to upload big files via Amazon S3.

Output file

The API will return a ZIP archive of .CSV files, one for each input file. The CSV files comply with our [CSV specification](#).

[Help](#)

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.

The screenshot shows the RebaseData website interface. At the top is the logo 'RebaseData' with three yellow circular icons. Below the logo are four main sections: 'MS Access converters', 'SQLite converters', 'MS SQL Server converters', and 'Firebird and Interbase converters'. Each section has a list of conversion options.

- MS Access converters:**
 - Convert Access to CSV
 - Convert Access to Excel
 - Convert Access to XLS
 - Convert Access to MySQL
 - Convert Access to MariaDB
 - Convert Access to SQLite
 - Convert Access to PSQL
 - Convert Access to PGSQL
 - Convert Access to Postgres
 - Convert Access to PostgreSQL
 - Convert Microsoft Access MDB to CSV
 - Convert Microsoft Access MDB to Excel
 - Convert Microsoft Access MDB to XLS
 - Convert Microsoft Access MDB to MySQL
 - Convert Microsoft Access MDB to MariaDB
 - Convert Microsoft Access MDB to SQLite
 - Convert Microsoft Access MDB to PSQL
 - Convert Microsoft Access MDB to PGSQL
 - Convert Microsoft Access MDB to Postgres
 - Convert Microsoft Access MDB to PostgreSQL
- SQLite converters:**
 - Convert SQLite to CSV
 - Convert SQLite to Excel
 - Convert SQLite to XLSX
 - Convert SQLite to MySQL
 - Convert SQLite to MariaDB
 - Convert SQLite to PSQL
 - Convert SQLite to PGSQL
 - Convert SQLite to Postgres
 - Convert SQLite to PostgreSQL
- MS SQL Server converters:**
 - Convert SQL Server MDF to CSV
 - Convert SQL Server MDF to Excel
 - Convert SQL Server MDF to XLS
 - Convert SQL Server MDF to XLSX
 - Convert SQL Server MDF to MySQL
 - Convert SQL Server MDF to MariaDB
 - Convert SQL Server MDF to SQLite
 - Convert SQL Server MDF to PSQL
 - Convert SQL Server MDF to PGSQL
 - Convert SQL Server MDF to Postgres
 - Convert SQL Server MDF to PostgreSQL
 - Convert SQL Server BAK to CSV
 - Convert SQL Server BAK to Excel
 - Convert SQL Server BAK to XLS
 - Convert SQL Server BAK to XLSX
 - Convert SQL Server BAK to MySQL
 - Convert SQL Server BAK to MariaDB
 - Convert SQL Server BAK to SQLite
 - Convert SQL Server BAK to PSQL
 - Convert SQL Server BAK to PGSQL
- Firebird and Interbase converters:**
 - Convert Firebird to CSV
 - Convert Firebird to Excel
 - Convert Firebird to XLSX
 - Convert Firebird to MySQL
 - Convert Firebird to MariaDB
 - Convert Firebird to SQLite
 - Convert Firebird to PSQL
 - Convert Firebird to PGSQL
 - Convert Firebird to Postgres
 - Convert Firebird to PostgreSQL
 - Convert FDB database to CSV
 - Convert FDB database to Excel
 - Convert FDB database to XLSX
 - Convert FDB database to MySQL
 - Convert FDB database to MariaDB
 - Convert FDB database to SQLite
 - Convert FDB database to PSQL
 - Convert FDB database to PGSQL
 - Convert FDB database to Postgres
 - Convert FDB database to PostgreSQL

They can also be converted to CSV using online converters:

[Rebasedata.com](http://rebasedata.com)

[Anyconv.com](http://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->duckduckgo_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->duckduckgo_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:

```

ddg_search.py
Day_9 > ddg_search.py
1  from duckduckgo_search import ddg
2
3  keywords = 'osint'
4  results = ddg(keywords, region='us-en', safesearch='Off', time='y')
5  print(results)
6

gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_9
gitpod /workspace/python-for-OSINT-21-days/Day_9 $ python ddg_search.py
[{"title": "What is Open-Source Intelligence (OSINT)? - SANS Institute", "href": "https://www.sans.org/blog/what-is-open-source-intelligence/", "body": "Open Source Intelligence (OSINT) is the collection, analysis, and dissemination of information that is publicly available and legally accessible. Right now, OSINT is used by organizations, including governments, businesses, and non-governmental organizations. It is useful in information gathering for a wide range of topics such as security ..."}, {"title": "9 Best OSINT Tools for 2023 (Paid & Free) - Comparitech", "href": "https://www.comparitech.com/net-admin/osint-tools/1", "body": "Here's our list of the eight best OSINT tools: OSINT Framework, a website directory of data discovery and gathering tool s for web and cloud platforms. You can understand what it uses to create reports for any search query, and it's a cloud-based service."}, {"title": "Open Source Intelligence (OSINT) Fundamentals - NICECS", "href": "https://nics.cisa.gov/education-training/catalog/cybersecurity/open-source-intelligence-osint-fundamentals", "body": "Open Source Intelligence (OSINT) Fundamentals. Online, Self-Paced. In this Open Source Intelligence (OSINT) Fundamentals training course, you will gain fundamental knowledge about OSINT, who uses it, and the ethical implications of using it. Upon completion, students will have a solid understanding of OSINT."}, {"title": "OSINT Penetration testing | Open-Source Intelligence Tools & Techniques", "href": "https://www.munculus.com/cybersecurity/penetration-testing-open-source-intelligence-tools/1", "body": "Starting with Open-Source Intelligence (OSINT): Tips, Tools, and Techniques. According to a 2021 IBM report, the average organization did not detect a data breach for up to 212 days and they did not fully contain the issue for another 75. In many instances, malicious hackers attack a company using publicly available information open-source intelligence, often referred to as OSINT."}, {"title": "OSINT Goes Mainstream: How Security Teams Can Use Open-Source ..", "href": "https://www.sans.org/cyber-security-career/white-papers/osint-goes-mainstream-how-security-teams-can-use-open-source-information-to-protect-companies-understand-risk/", "body": "According to the Office of the U.S. Director of National Intelligence, open-source intelligence, also known as OSINT, is defined as \"publicly available information appearing in print or ...\""}, {"title": "Open Source Intelligence: The Beginners' Guide to OSINT", "href": "https://www.liferaffinc.com/blog/the-beginners-guide-to-osint", "body": "Open source intelligence, or OSINT, refers to any information collected from free public sources about an organization or individual to provide actionable insights for decision-makers. Technically, this information could be anything from news articles to social media posts."}

```

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:

```
ddg_search_to_csv.py
Day_9 > ddg_search_to_csv.py
1   from duckduckgo_search import ddg
2   import csv
3
4   csv_file = open('search_results.csv', 'w')
5   writer = csv.writer(csv_file, delimiter=';')
6
7
8   keywords = 'osint'
9   results = ddg(keywords, region='us-en', safesearch='Off', time='y')
10
11  for x in range(len(results)):
12      row = [results[x]["title"], results[x]["body"], results[x]["href"]]
13      writer.writerow(row)
14
15  csv_file.close()
16
```

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL

```
gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_9
gitpod /workspace/python-for-OSINT-21-days/Day_9 (main) $ python ddg_search_to_csv.py
gitpod /workspace/python-for-OSINT-21-days/Day_9 (main) $ 
```

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)
```

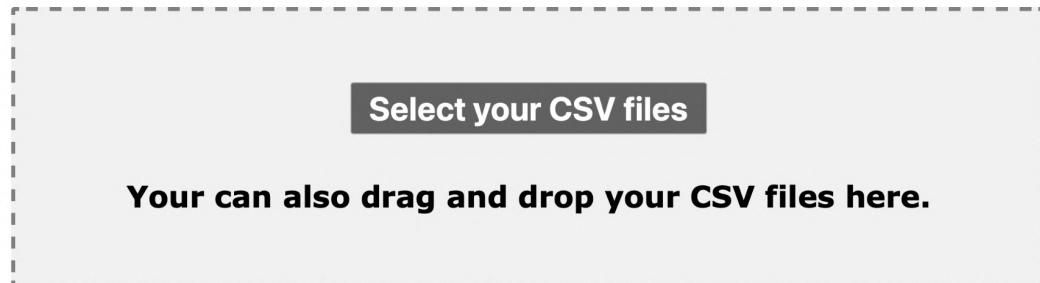
Close search_results.csv file:

```
csv_file.close()
```

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

| | | search_results |
|----|--|--|
| 1 | What is OSINT (Open-Source Intelligence?) SANS Institute | Open Source Intelligence (OSINT) is the co |
| 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 & Article | 4) OSINT Tool: ScamSearch.io - A Global Data |
| 4 | Spy agencies look to standardize use of open source intelligence | The Defense Intelligence Agency is also looki |
| 5 | What Is Open Source Intelligence: The Importance of OSINT in 2022 | Open-source intelligence, or OSINT, refers to |
| 6 | What is OSINT? Defining open-source intelligence in 2022 - Social Links | To start at the beginning, OSINT is an acronym |
| 7 | Types of Intelligence Collection - Intelligence Studies - LibGuides | Although HUMINT is an important collection o |
| 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防ぐ方法 | OSINTはサイバー攻撃者の特定や個人情報流 |
| 9 | Open-Source Intelligence (OSINT) - Cyber | Open-Source Intelligence (OSINT) Apakah ka |
| 10 | Ukraine OSINT.org | Unveiling the Power of VPNs: How OSINT Pr |
| 11 | Open Source Intelligence: The Beginners' Guide to OSINT | Open source intelligence, or OSINT, refers to |
| 12 | Separating OSINT from the Secret World Strengthens Both | These arguments are serious and compelling. |
| 13 | What is OSINT? Open-Source Intelligence Tools 2023 Gridinsoft | December 07, 2022. OSINT, or Open-Source |
| 14 | 12 Resourceful OSINT Tools You Should Know - MUO | Open-source intelligence (OSINT) is the proc |
| 15 | What is Open Source Intelligence, and how is it used? - News & Article | OSINT is used for threat intelligence, which is |
| 16 | Open-Source Intelligence is Indispensable for Countering Threats | In brief, OSINT is the painstaking gathering a |
| 17 | GitHub - Datalux/Osintgram: Osintgram is a OSINT tool on Instagram | Osintgram is a OSINT tool on Instagram to co |
| 18 | 9 Open Source Intelligence (OSINT) Tools for Penetration Testing | 9 Open Source Intelligence (OSINT) Tools for |
| 19 | OSINT Goes Mainstream: How Security Teams Can Use Open-Source Intelligence | According to the Office of the U.S. Director of |
| 20 | Penetration Testing with Open-Source Intelligence (OSINT): Tips and Techniques | Penetration Testing with Open-Source Intellig |
| 21 | What is OSINT Open Source Intelligence? CrowdStrike | The OSINT framework is a methodology that |

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:

The screenshot shows a code editor interface with several files listed in the Explorer sidebar, including `ddg_search_to_csv.py`, `ddg_search_download_pdf.py`, and multiple files under `PYTHON-FOR-OSINT-21-DAYS`. The terminal window at the bottom shows the command `gitpod /workspace/python-for-osint-21-days (main) $ cd Day_9` and the execution of `python ddg_search_download_pdf.py`. The output indicates that 20 documents are being downloaded, with a progress bar showing 100% completion.

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 also collect Answers, Images, Videos, News, Maps, Suggestions, and translate text.

DuckDuckGo Help Pages

Search help pages...

- Welcome
- Add-Ons
- Community
- Company
- Desktop
- Features
- Mobile
- Open-Source
- App Tracking Protection

Search Operators

| Example | Result |
|-------------------------|---|
| cats dogs | Results about cats or dogs |
| "cats and dogs" | Results for exact term "cats and dogs". If no results are found, we'll try to show related results. |
| cats -dogs | Fewer dogs in results |
| cats +dogs | More dogs in results |
| cats filetype:pdf | PDFs about cats. Supported file types: pdf, doc(x), xls(x), ppt(x), html |
| dogs site:example.com | Pages about dogs from example.com |
| cats - site:example.com | Pages about cats, excluding example.com |
| intitle:dogs | Page title includes the word "dogs" |
| inurl:cats | Page url includes the word "cats" |

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/jnhgnonknehpejnehehllkliplmbmhn>)

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/bjkpgfhekfmddphnniobddhkljmmlj>

ScrapeStorm

<https://www.scrapestorm.com>

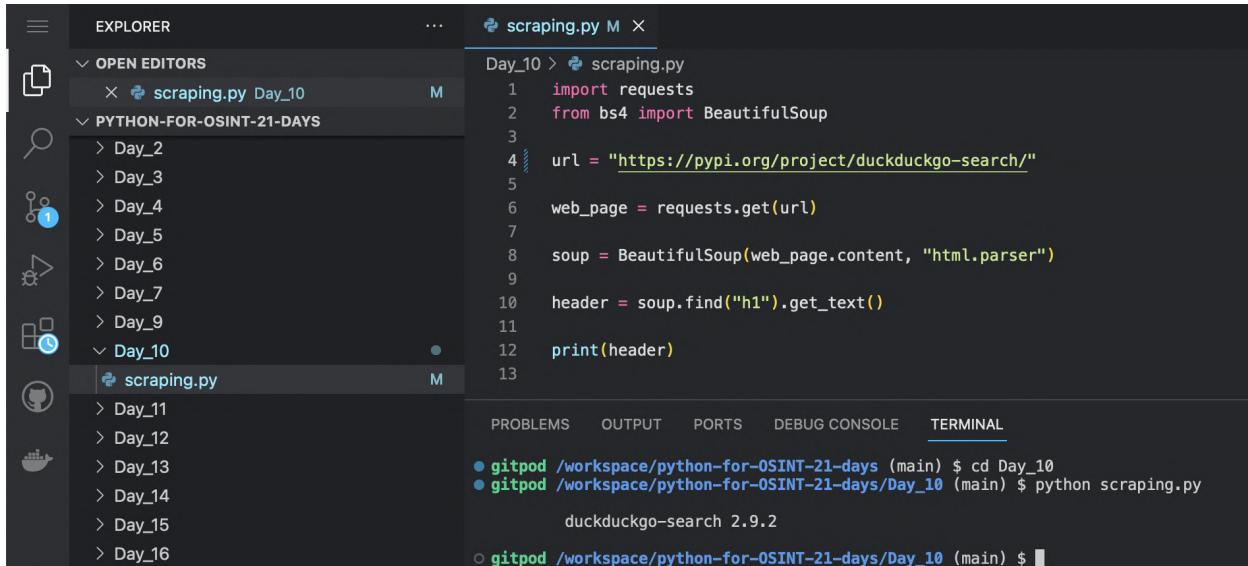
And almost every day some new AI scraping tool appears.

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>).



```
Day_10 > scraping.py
1 import requests
2 from bs4 import BeautifulSoup
3
4 url = "https://pypi.org/project/duckduckgo-search/"
5
6 web_page = requests.get(url)
7
8 soup = BeautifulSoup(web_page.content, "html.parser")
9
10 header = soup.find("h1").get_text()
11
12 print(header)
13
```

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL

- gitpod /workspace/python-for-OSINT-21-days (main) \$ cd Day_10
- gitpod /workspace/python-for-OSINT-21-days/Day_10 (main) \$ python scraping.py
- gitpod /workspace/python-for-OSINT-21-days/Day_10 (main) \$ duckduckgo-search 2.9.2

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.

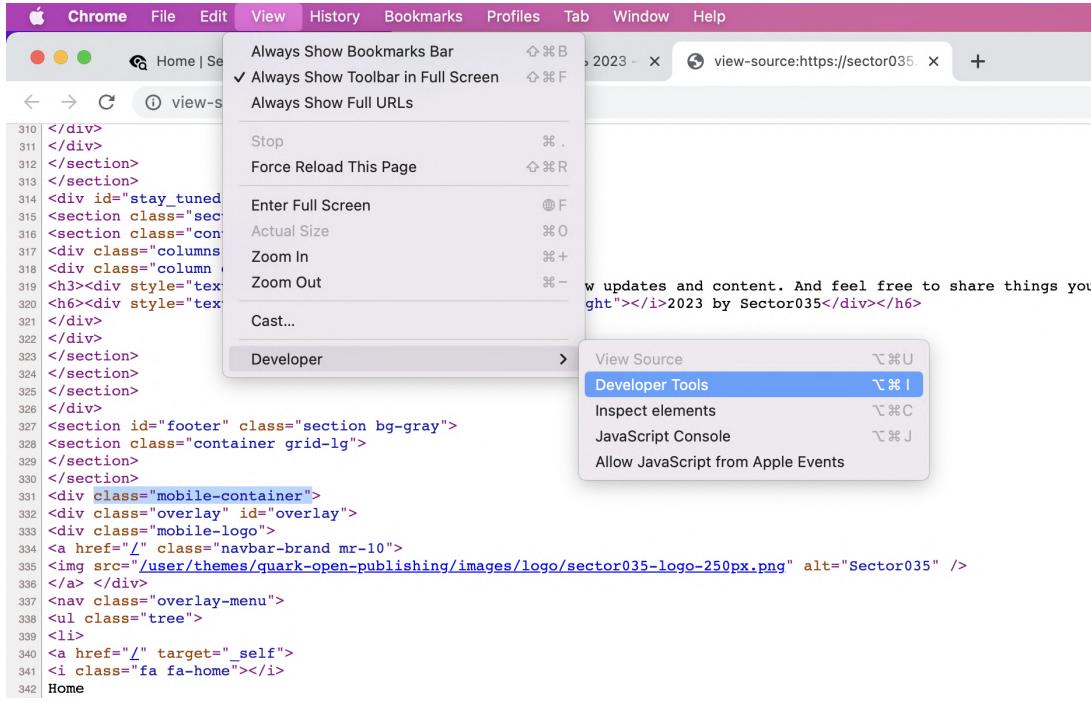
“div.redblocks” - elements with div tag and redblocks class.

“[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](#)

SectorQ35

[Home](#) [Week in OSINT](#) [Articles](#) [Links](#) [Search](#)



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.

| Raw <i>i</i> | Rendered <i>i</i> | Difference <i>i</i> (720 of 2272) |
|--|---|--|
|  |  |  |

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).

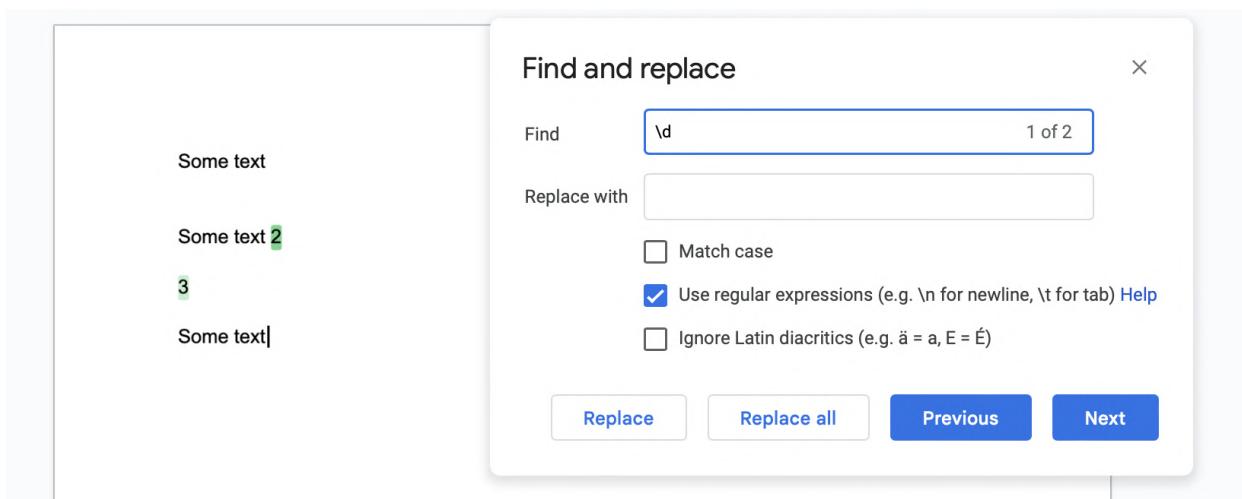
For scraping websites where the code changes a lot after the execution of JavaScript code in the browser, you can use such packages as Selenium (<https://selenium-python.readthedocs.io>). It allows you to use Python to open different browsers and simulate user actions in them.

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.

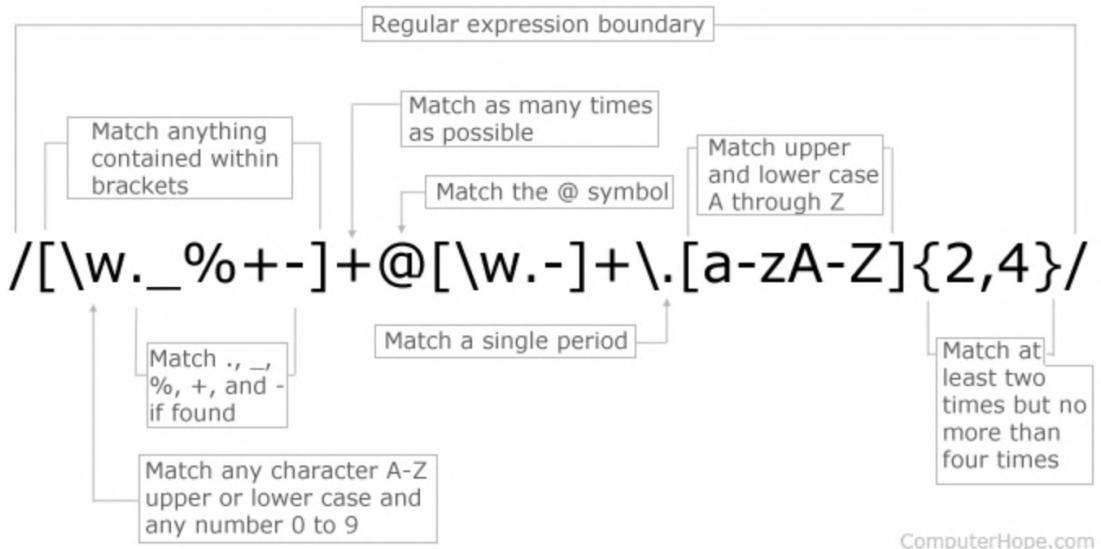
Regular expressions are supported by most popular programming languages and are used for searching, validating and extracting different data. For example, phone numbers, email addresses, IP addresses, cryptocurrency wallet numbers, etc.

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:

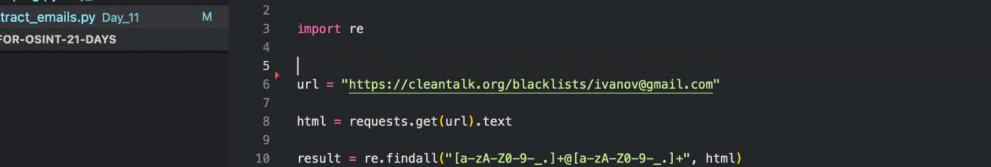
Regular Expression E-mail Matching Example



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:



```
gitpod /workspace/python-for-OSINT-21-days $ cd Day_11
gitpod /workspace/python-for-OSINT-21-days$ Day_11 $ python 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.

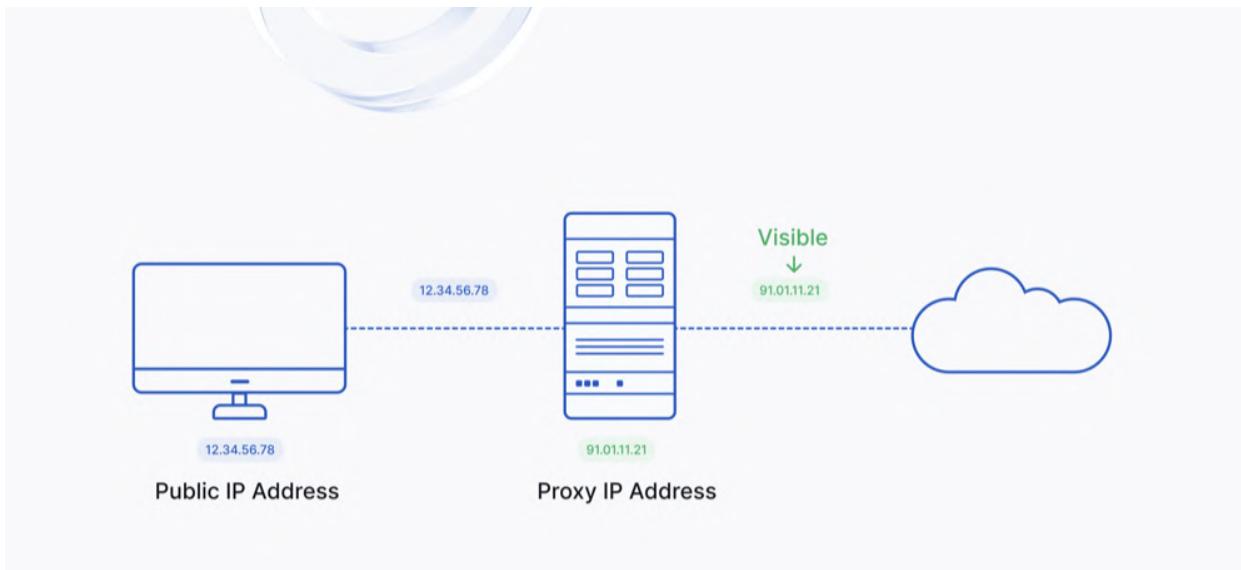
There is only one code example in this lesson, since your main goal today is to learn in detail how regular expressions are used in OSINT.

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).

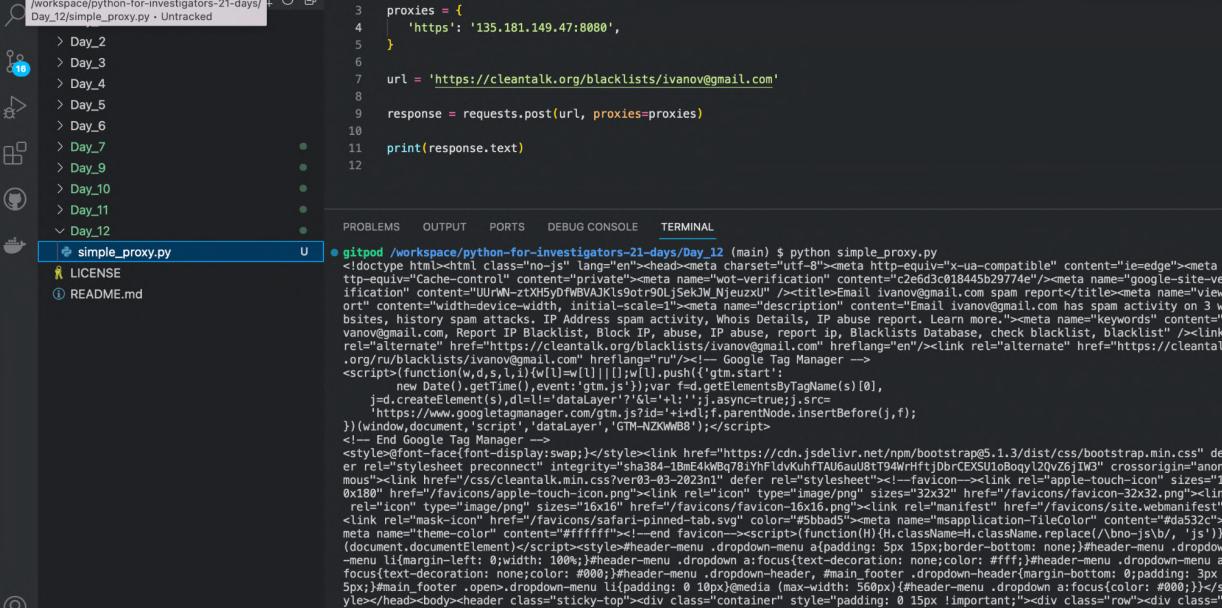


Thanks for picture, [Upguard.com](https://upguard.com/).

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:



```
simple_proxy.py U x
Day_12 > simple_proxy.py
1 import requests
2
3 proxies = {
4     'https': '135.181.149.47:8080',
5 }
6
7 url = 'https://cleantalk.org/blacklists/ivanov@gmail.com'
8
9 response = requests.post(url, proxies=proxies)
10
11 print(response.text)
12
```

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL

```
gitpod /workspace/python-for-investigators-21-days/Day_12 (main) $ python simple_proxy.py
<!DOCTYPE html><html class="no-js" lang="en"><head><meta charset="utf-8"><meta http-equiv="x-ua-compatible" content="ie=edge"><meta name="viewport" content="width=device-width, initial-scale=1"><meta name="description" content="Email ivanov@gmail.com has spam activity on 3 websites, history spam attacks. IP Address spam activity, Whois Details, IP abuse report. Learn more."><meta name="keywords" content="ivanov@gmail.com, Report IP Blacklist, Block IP, abuse, report ip, Blacklists Database, check blacklist, blacklist" /><link rel="alternate" href="https://cleantalk.org/blacklists/ivanov@gmail.com" hreflang="en"/><link rel="alternate" href="https://cleantalk.org/blacklists/ivanov@gmail.com" hreflang="ru"/><!-- Google Tag Manager -->
<script>(function(w,d,s,l,i){w[l]=w[l]||[];w[l].push({'gtm.start':
    new Date().getTime(),event:'gtm.js'});var f=d.getElementsByTagNamebyTagName(s)[0],
    j=d.createElement(s),l=!l;j.async=true;j.src=
    'https://www.googletagmanager.com/gtm.js?id='+i+";f.parentNode.insertBefore(j,f);
})(window,document,'script','dataLayer','GTM-NZKWB8');
```

```
# 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.

The screenshot shows the hidemy.name website interface for finding proxies. At the top, there's a navigation bar with links for 'What is a VPN?', 'Pricing', 'Download', 'Help', and a 'Buy access' button. Below the navigation is a search form with dropdowns for 'Country' (set to 'All country (81)'), 'Proxy speed' (set to 'ms'), and checkboxes for 'Proxy types' (HTTP is checked) and 'Anonymity' (High is checked). There's also a field for 'Port number' with a placeholder 'can be separated by commas'. Below the search form are buttons for 'Show' (in blue), 'Paid Features' (in white), and download links for 'Export IP:Port or Excel'. A note below the search form says 'Don't know how to use a proxy? Check the instructions for your browser' with icons for various browsers. At the bottom is a table showing a list of proxy servers with columns for IP address, Port, Country/City, Speed, Type, Anonymity, and Latest update.

| IP address | Port | Country, City | Speed | Type | Anonymity | Latest update |
|----------------|------|-----------------------|--------|------|-----------|---------------|
| 172.67.8.142 | 80 | United States | 160 ms | HTTP | no | 2 minutes |
| 191.101.251.42 | 80 | Netherlands Rotterdam | 580 ms | HTTP | no | 2 minutes |
| 45.8.105.111 | 80 | Curacao | 600 ms | HTTP | no | 2 minutes |

Examples of proxy servers list:

<https://hidemy.name/en/>

<https://github.com/clarktm/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 permutation.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](#)

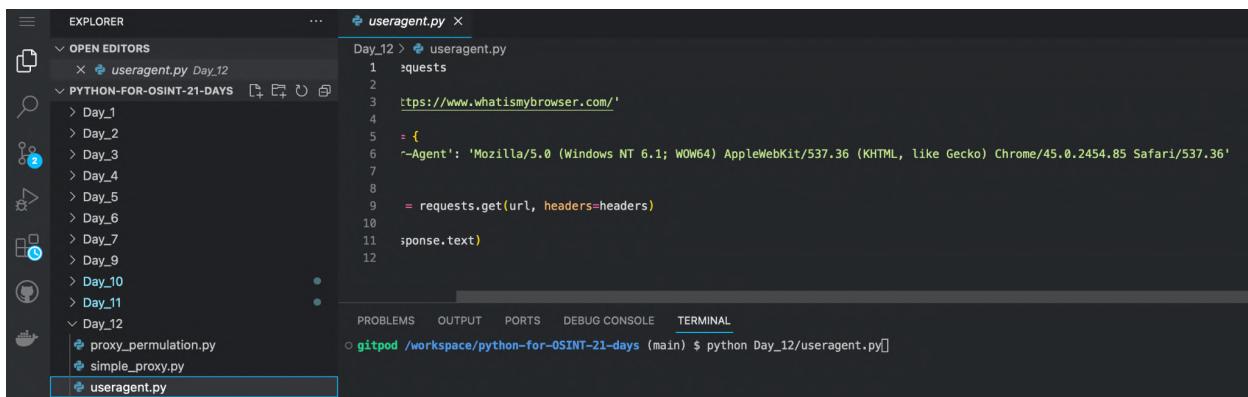
[mitmproxy](#)

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:



```
useragent.py
1 import requests
2
3 url = 'https://www.whatismybrowser.com/'
4
5 headers = {
6     'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.85 Safari/537.36'
7 }
8
9 response = requests.get(url, headers=headers)
10
11 print(response.text)
```

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):

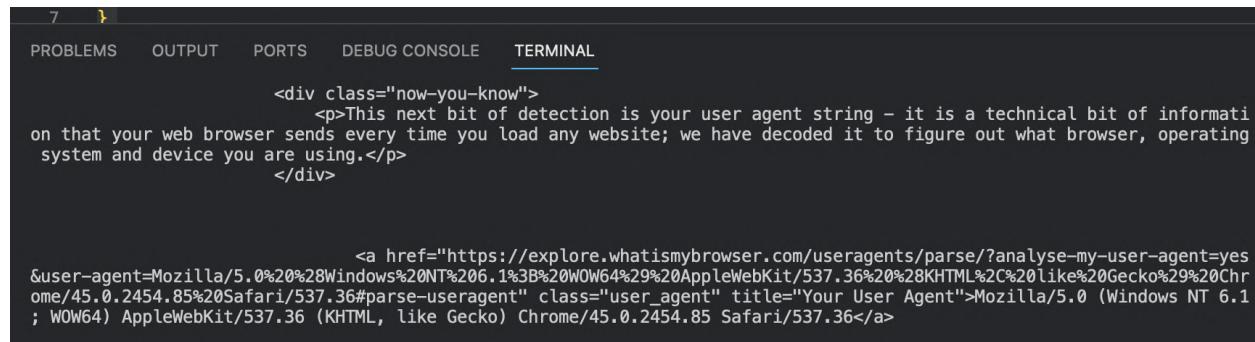
```
headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.85 Safari/537.36'
}
```

```
# Send request:
```

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

```
# Print response text:
```

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



```
PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL

<div class="now-you-know">
    <p>This next bit of detection is your user agent string – it is a technical bit of information that your web browser sends every time you load any website; we have decoded it to figure out what browser, operating system and device you are using.</p>
</div>

<a href="https://explore.whatismybrowser.com/useragents/parse/?analyse-my-user-agent=yes&user-agent=Mozilla/5.0%20%28Windows%20NT%206.1%3B%20WOW64%29%20AppleWebKit/537.36%20%28KHTML%20like%20Gecko%29%20Chrome/45.0.2454.85%20Safari/537.36#parse-useragent" class="user_agent" title="Your User Agent">Mozilla/5.0 (Windows NT 6.1 ; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.85 Safari/537.36</a>
```

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.

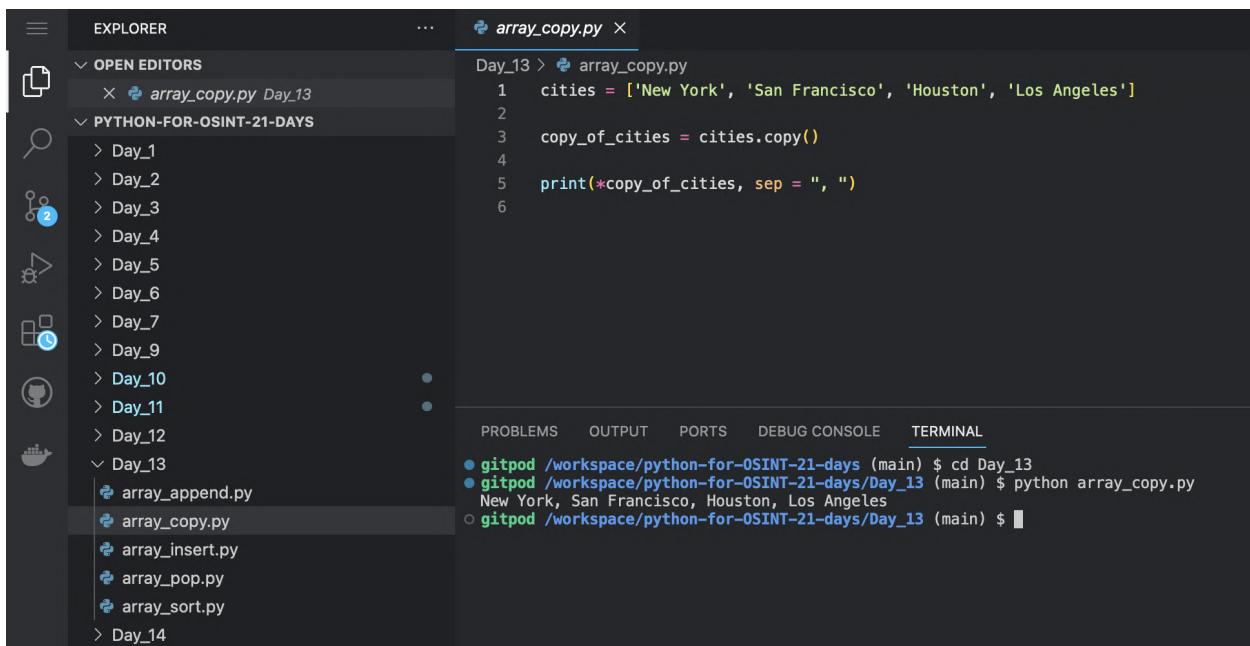
Day 13. Functions for working with lists

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:



```
array_copy.py
1 cities = ['New York', 'San Francisco', 'Houston', 'Los Angeles']
2
3 copy_of_cities = cities.copy()
4
5 print(*copy_of_cities, sep = ", ")
```

Create list of USA cities:

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

Copy list of USA cities:

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

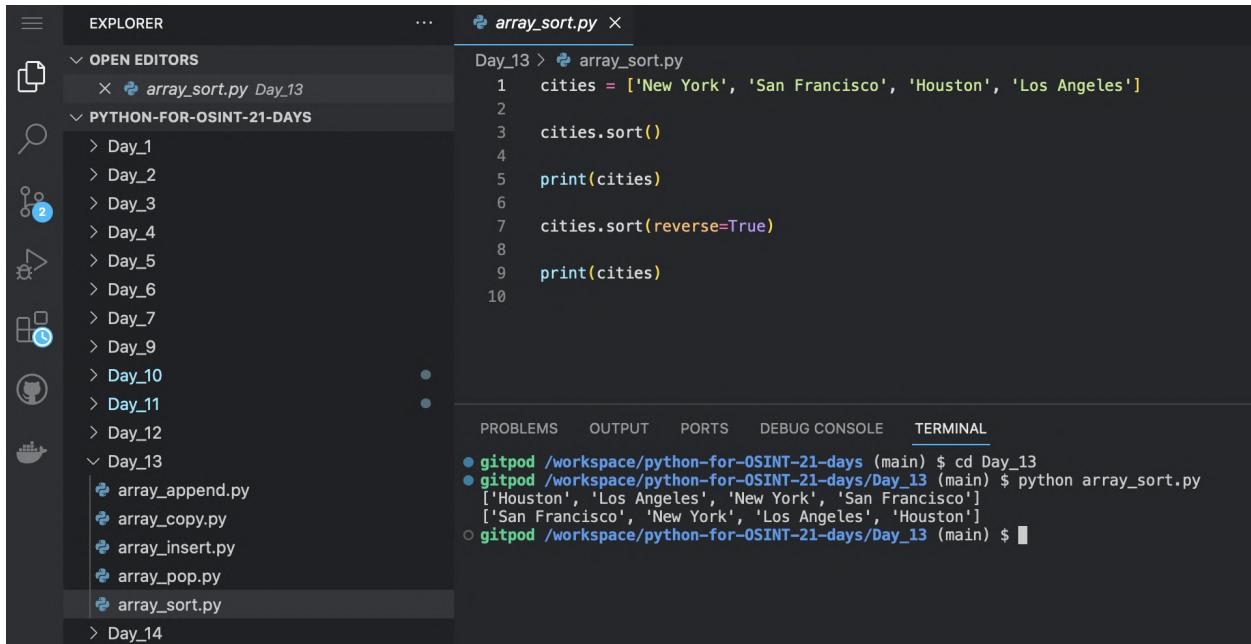
```
# Print copy of list of USA cities with elements separated by ",":
```

```
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:



```
array_sort.py
Day_13 > array_sort.py
1  cities = ['New York', 'San Francisco', 'Houston', 'Los Angeles']
2
3  cities.sort()
4
5  print(cities)
6
7  cities.sort(reverse=True)
8
9  print(cities)
10
```

The screenshot shows the VS Code interface. The left sidebar has icons for file operations, search, and workspace navigation. The main area shows the file structure under 'EXPLORER' and the content of 'array_sort.py'. Below the code editor is a tab bar with 'PROBLEMS', 'OUTPUT', 'PORTS', 'DEBUG CONSOLE', and 'TERMINAL'. The 'TERMINAL' tab is active, displaying command-line history:

- gitpod /workspace/python-for-OSINT-21-days (main) \$ cd Day_13
- gitpod /workspace/python-for-OSINT-21-days/Day_13 (main) \$ python array_sort.py
- ['Houston', 'Los Angeles', 'New York', 'San Francisco']
- ['San Francisco', 'New York', 'Los Angeles', 'Houston']
- gitpod /workspace/python-for-OSINT-21-days/Day_13 (main) \$

```
# 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 :

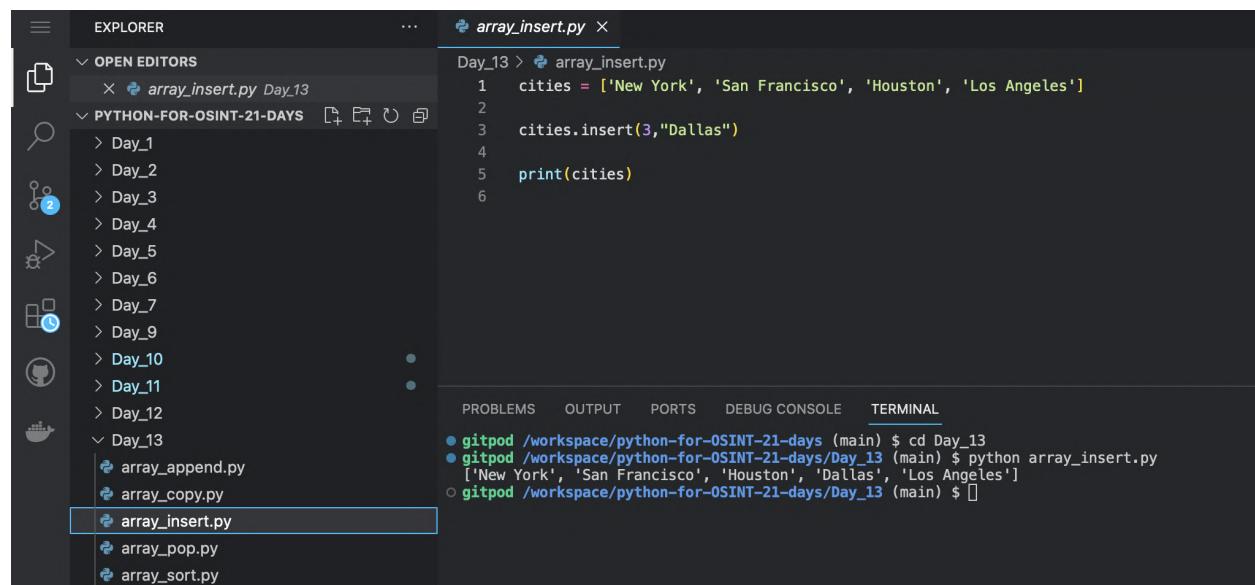
```
cities.sort(reverse=True)
```

Print results :

```
print(cities)
```

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

Run array_insert.py:



```
array_insert.py
Day_13 > array_insert.py
1   cities = ['New York', 'San Francisco', 'Houston', 'Los Angeles']
2
3   cities.insert(3,"Dallas")
4
5   print(cities)
6

PROBLEMS    OUTPUT    PORTS    DEBUG CONSOLE    TERMINAL
● gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_13
● gitpod /workspace/python-for-OSINT-21-days/Day_13 (main) $ python array_insert.py
['New York', 'San Francisco', 'Houston', 'Dallas', 'Los Angeles']
○ gitpod /workspace/python-for-OSINT-21-days/Day_13 (main) $ 
```

Create a list of U.S. city names:

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

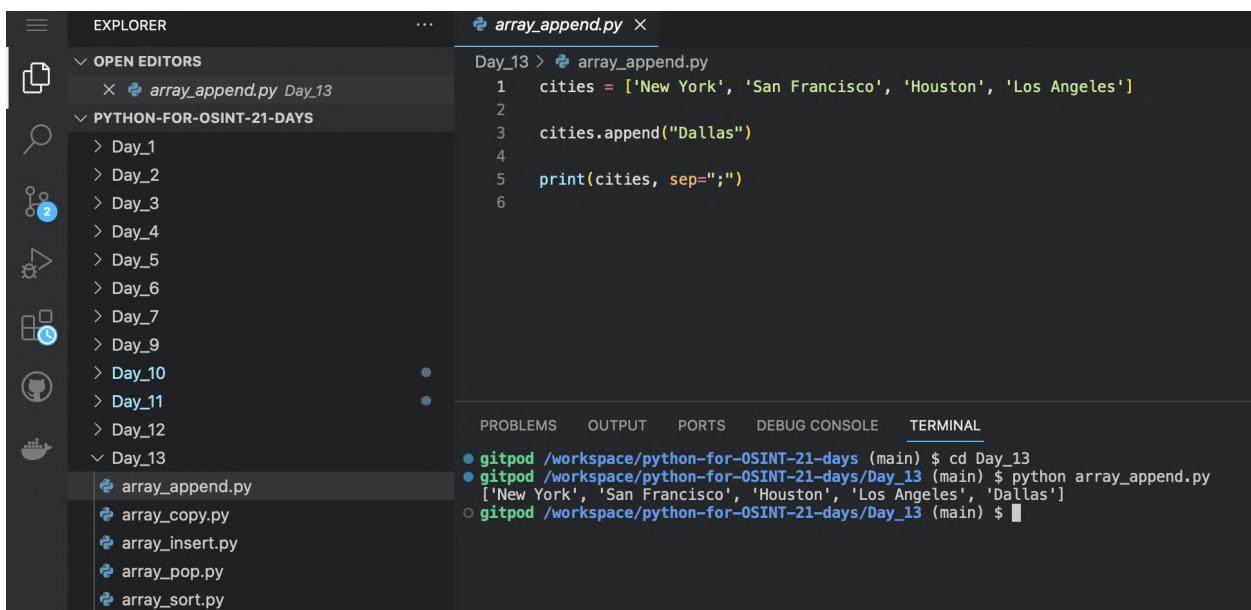
Add item number three and the text Dallas (remember that the count starts with zero):

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

Display the modified list on the screen:

```
print(cities)
```

Run array_append.py:



```
array_append.py
Day_13 > array_append.py
1  cities = ['New York', 'San Francisco', 'Houston', 'Los Angeles']
2
3  cities.append("Dallas")
4
5  print(cities, sep=";")
6
```

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL

- gitpod /workspace/python-for-OSINT-21-days (main) \$ cd Day_13
- gitpod /workspace/python-for-OSINT-21-days/Day_13 (main) \$ python array_append.py ['New York', 'San Francisco', 'Houston', 'Los Angeles', 'Dallas']
- gitpod /workspace/python-for-OSINT-21-days/Day_13 (main) \$

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")
```

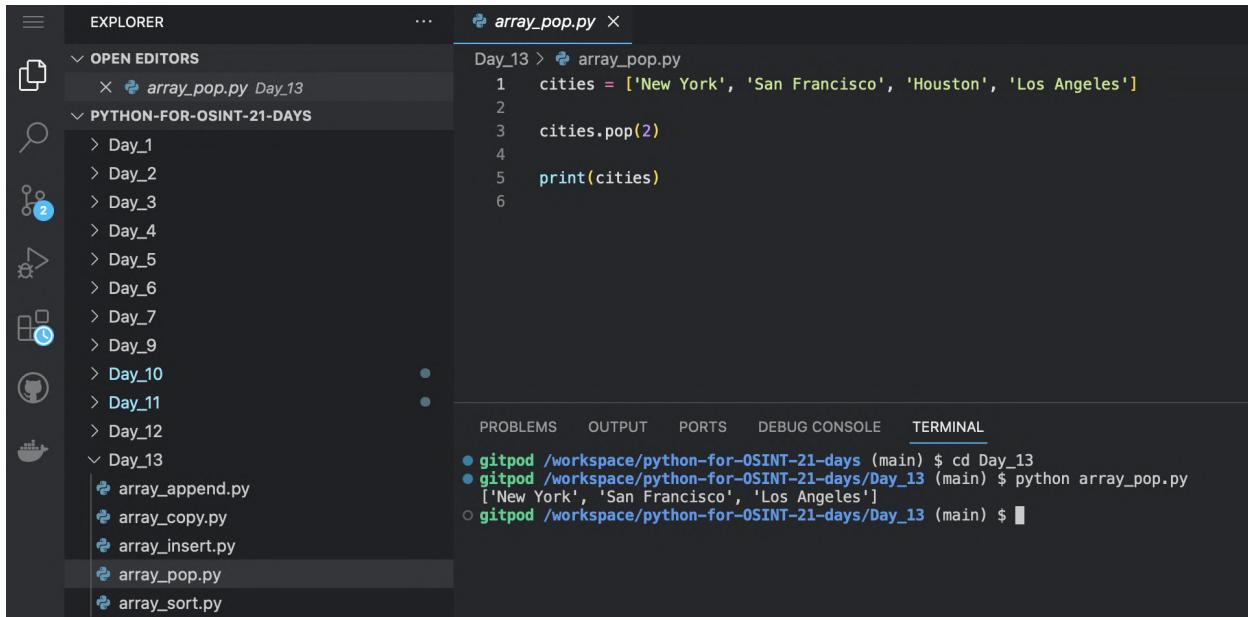
Display the list on the screen, separating elements with semicolons:

```
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`:



```
array_pop.py
Day_13 > array_pop.py
1 cities = ['New York', 'San Francisco', 'Houston', 'Los Angeles']
2
3 cities.pop(2)
4
5 print(cities)
6
```

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL

- gitpod /workspace/python-for-OSINT-21-days (main) \$ cd Day_13
- gitpod /workspace/python-for-OSINT-21-days/Day_13 (main) \$ python array_pop.py
- ['New York', 'San Francisco', 'Los Angeles']
- gitpod /workspace/python-for-OSINT-21-days/Day_13 (main) \$

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.

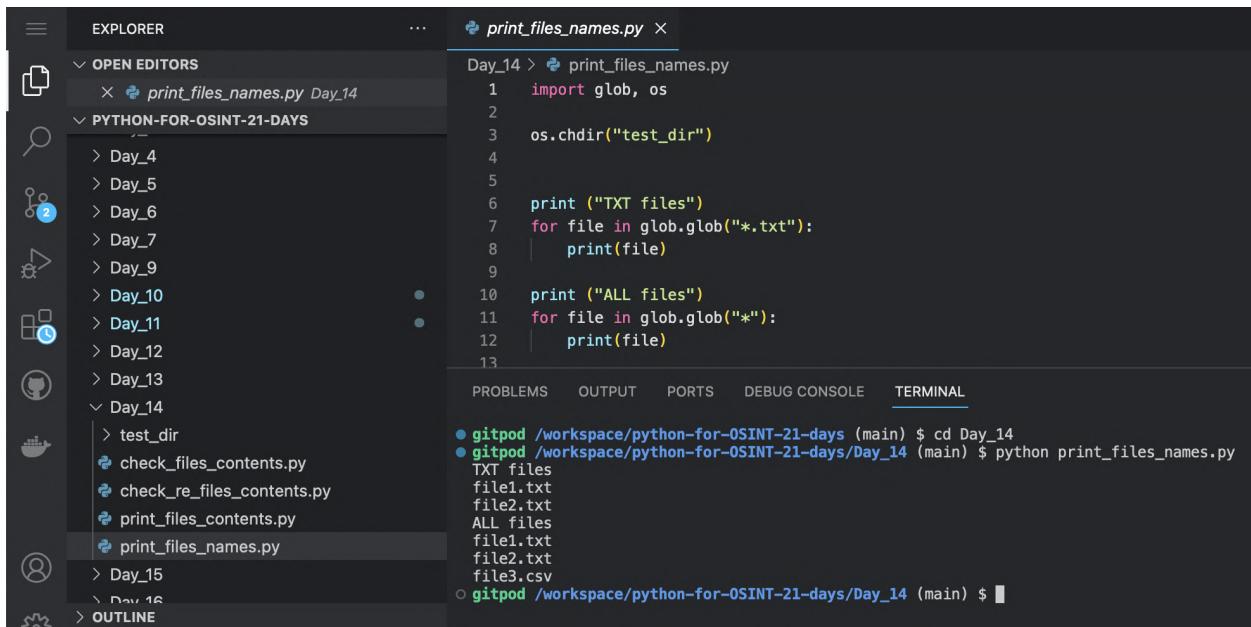
Day 14. Working with the file system

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:



The screenshot shows a code editor interface with the following details:

- EXPLORER:** Shows a tree view of a workspace named "PYTHON-FOR-OSINT-21-DAYS". Inside this folder are subfolders "Day_4" through "Day_16" and several Python files: "check_files_contents.py", "check_re_files_contents.py", "print_files_contents.py", and "print_files_names.py".
- print_files_names.py:** The code in this file is:

```
import glob, os
os.chdir("test_dir")
print ("TXT files")
for file in glob.glob("*.txt"):
    print(file)
print ("ALL files")
for file in glob.glob("*"):
    print(file)
```
- TERMINAL:** Shows the command-line output of running the script:

```
gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_14
gitpod /workspace/python-for-OSINT-21-days/Day_14 (main) $ python print_files_names.py
TXT files
file1.txt
file2.txt
ALL files
file1.txt
file2.txt
file3.csv
```

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`:

```
print_files_contents.py
Day_14 > print_files_contents.py
1 import glob, os
2
3 os.chdir("test_dir")
4
5 for file in glob.glob("*.txt"):
6
7     with open (file) as current_file:
8         print(current_file.readlines())
9
```

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL

```
gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_14
gitpod /workspace/python-for-OSINT-21-days/Day_14 (main) $ python print_files_contents.py
['Hello from file1.txt!\n']
['Hello from file2.txt!\n']
gitpod /workspace/python-for-OSINT-21-days/Day_14 (main) $
```

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:

```
check_files_contents.py
Day_14 > check_files_contents.py Day_14
1 import glob, os
2
3 os.chdir("test_dir")
4
5 for file in glob.glob("*"):
6
7     with open (file) as current_file:
8         current_file_content=current_file.read()
9         if "2" in current_file_content:
10             print(current_file_content)

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL
gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_14
gitpod /workspace/python-for-OSINT-21-days/Day_14 (main) $ python check_files_contents.py
Hello from file2.txt!
gitpod /workspace/python-for-OSINT-21-days/Day_14 (main) $
```

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()
```

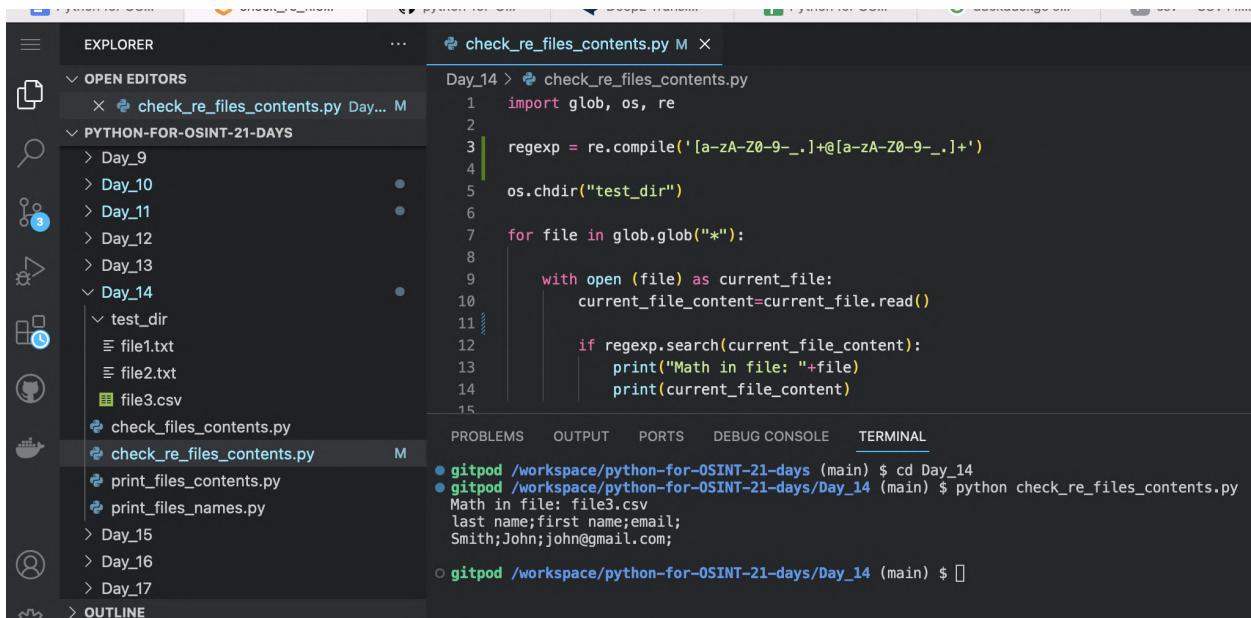
If file content include “2”, print it:

```
            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`:



The screenshot shows the VS Code interface. The left sidebar displays a file tree with a folder named 'Day_14' containing several Python files: 'check_re_files_contents.py', 'check_files_contents.py', 'print_files_contents.py', and 'print_files_names.py'. Below these are sub-folders 'Day_9' through 'Day_17'. Inside 'Day_14', there is a 'test_dir' folder containing three files: 'file1.txt', 'file2.txt', and 'file3.csv'. The main code editor window shows the content of 'check_re_files_contents.py'. The terminal window at the bottom shows the command being run: 'gitpod /workspace/python-for-OSINT-21-days (main) \$ cd Day_14' followed by 'gitpod /workspace/python-for-OSINT-21-days/Day_14 (main) \$ python check_re_files_contents.py'. The output of the script is displayed, showing it found an email address in 'file3.csv': 'Math in file: file3.csv' and 'last name;first name;email; Smith;John;john@gmail.com;'.

```
check_re_files_contents.py M
Day_14 > check_re_files_contents.py
1 import glob, os, re
2
3 regexp = re.compile('[a-zA-Z0-9-.]+@[a-zA-Z0-9-.]+')
4
5 os.chdir("test_dir")
6
7 for file in glob.glob("*"):
8
9     with open(file) as current_file:
10         current_file_content=current_file.read()
11
12         if regexp.search(current_file_content):
13             print("Math in file: "+file)
14             print(current_file_content)
15
```

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL

```
gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_14
gitpod /workspace/python-for-OSINT-21-days/Day_14 (main) $ python check_re_files_contents.py
Math in file: file3.csv
last name;first name;email;
Smith;John;john@gmail.com;

gitpod /workspace/python-for-OSINT-21-days/Day_14 (main) $
```

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](#))

The screenshot shows the who.is website interface. At the top, there is a search bar with 'nytimes.com' and a magnifying glass icon. To the right of the search bar are links for 'Premium Domains', 'Transfer', and 'Features'. Below the search bar, the page displays the WHOIS data for the domain 'nytimes.com' under three main sections: 'Registrant Contact Information', 'Administrative Contact Information', and 'Technical Contact Information'. The data is presented in a table-like format with 'Name' and 'Organization' columns.

| Registrant Contact Information: | |
|---------------------------------|--|
| Name | Domain Administrator |
| Organization | The New York Times Company |
| Address | 620 8th Avenue, |
| City | New York |
| State / Province | NY |
| Postal Code | 10018 |
| Country | US |
| Phone | +1.2125561234 |
| Fax | +1.2125561234 |
| Email | hostmaster@nytimes.com |

| Administrative Contact Information: | |
|-------------------------------------|--|
| Name | Domain Administrator |
| Organization | The New York Times Company |
| Address | 620 8th Avenue, |
| City | New York |
| State / Province | NY |
| Postal Code | 10018 |
| Country | US |
| Phone | +1.2125561234 |
| Fax | +1.2125561234 |
| Email | hostmaster@nytimes.com |

| Technical Contact Information: | |
|--------------------------------|----------------------------|
| Name | Domain Administrator |
| Organization | The New York Times Company |

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/>).

```
● gitpod /workspace/python-for-OSINT-21-days (main) $ pip install python-whois
Collecting python-whois
  Downloading python-whois-0.8.0.tar.gz (109 kB)
    ━━━━━━━━━━━━━━━━ 109.6/109.6 kB 4.0 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting future
  Downloading future-0.18.3.tar.gz (840 kB)
    ━━━━━━━━━━━━━━ 840.9/840.9 kB 12.1 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: python-whois, future
  Building wheel for python-whois (setup.py) ... done
    Created wheel for python-whois: filename=python_whois-0.8.0-py3-none-any.whl size=103247 sha256=d2f3d8b0722183149ad48a0a35b57122da35686aac924c9a78635a05a6021a5e
    Stored in directory: /workspace/.pyenv_mirror/pip_cache/wheels/8a/d4/1d/bab4b44ad52eadf1b10c5c1ec7cb18a936f24b58bfb95b427e
      Building wheel for future (setup.py) ... done
      Created wheel for future: filename=future-0.18.3-py3-none-any.whl size=492022 sha256=938ca2bd301dabaf994397a06137f50bed71d80e101a1728862e82ddbf6bdd4
      Stored in directory: /workspace/.pyenv_mirror/pip_cache/wheels/da/19/ca/9d8c44cd311a955509d7e13da3f0bea42400c469ef825b580b
  Successfully built python-whois future
  Installing collected packages: future, python-whois
  Successfully installed future-0.18.3 python-whois-0.8.0
```

pip install python-whois

Run `whois_info.py`:

```
Day_15 > whois_info.py
1 import whois
2
3 whois_info = whois.whois('sector035.nl')
4
5 print (whois_info)
6
7 print("Creation date")
8
9 print(whois_info["creation_date"])
10

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL

● gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_15
● gitpod /workspace/python-for-OSINT-21-days/Day_15 (main) $ python whois_info.py
{
  "domain_name": "sector035.nl",
  "expiration_date": null,
  "updated_date": "2017-12-28 00:00:00",
  "creation_date": "2017-05-29 00:00:00",
  "status": "active",
  "registrar": "Easyhosting B.V.",
  "registrar_address": "Kabelweg 21",
  "registrar_postal_code": "1014BA",
  "registrar_city": "Amsterdam",
  "registrar_country": "Netherlands",
  "dnssec": "no",
  "name_servers": [
    "becky.ns.cloudflare.com",
    "jonah.ns.cloudflare.com"
  ]
}
Creation date
2017-05-29 00:00:00
○ gitpod /workspace/python-for-OSINT-21-days/Day_15 (main) $
```

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](#))

The screenshot shows the SuperTool Beta7 interface. At the top, there is a search bar with 'nytimes.com' and a 'DNS Lookup' button. Below the search bar, the domain 'a:nytimes.com' is selected. A green button labeled 'Find Problems' is visible. The main content area displays a table of DNS records:

| Type | Domain Name | IP Address | TTL |
|------|-------------|---|---------|
| A | nytimes.com | 151.101.1.164 Fastly, Inc. (AS54113) | 120 sec |
| A | nytimes.com | 151.101.65.164 Fastly, Inc. (AS54113) | 120 sec |
| A | nytimes.com | 151.101.129.164 Fastly, Inc. (AS54113) | 120 sec |
| A | nytimes.com | 151.101.193.164 Fastly, Inc. (AS54113) | 120 sec |

Below the table, there is a 'Test' section with a row for 'DNS Record Published' showing 'DNS Record found'. A message at the bottom states 'Your DNS hosting provider is "NS1" Need Bulk Dns Provider Data?'. At the very bottom, there are links for 'dns check', 'mx lookup', 'dmarc lookup', 'spf lookup', 'dns propagation', and a 'Transcript' link.

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 dns
import dns.resolver

result = dns.resolver.resolve('osintme.com', 'A')
for ipval in result:
    print('IP', ipval.to_text())
```

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL

- gitpod /workspace/python-for-OSINT-21-days (main) \$ cd Day_15
- gitpod /workspace/python-for-OSINT-21-days/Day_15 (main) \$ python dns_info.py
- IP 2.57.138.1
- gitpod /workspace/python-for-OSINT-21-days/Day_15 (main) \$

Import python dns 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: autocfg.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
```

```
nytimes_subdomains.txt
1 We have 105 possibilities
2 Starting at 2023-05-04 14:35:14.041917
3
4 testing: 1.nytimes.com
5 testing: 1rer.nytimes.com
6 testing: 2.nytimes.com
7 testing: 2tty.nytimes.com
8 testing: 42.nytimes.com
9 testing: admin.nytimes.com
10 testing: api.nytimes.com
11 testing: app.nytimes.com
12 testing: apps.nytimes.com
13 testing: autoconfig.nytimes.com
14 testing: autodiscover.nytimes.com
15 testing: bbs.nytimes.com
16 testing: beta.nytimes.com
17 testing: blog.nytimes.com
18 testing: cdn.nytimes.com
19 testing: clients.nytimes.com
20 testing: cloud.nytimes.com
21 testing: code.nytimes.com
22 testing: customers.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/apisintDS | 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.apiry.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).

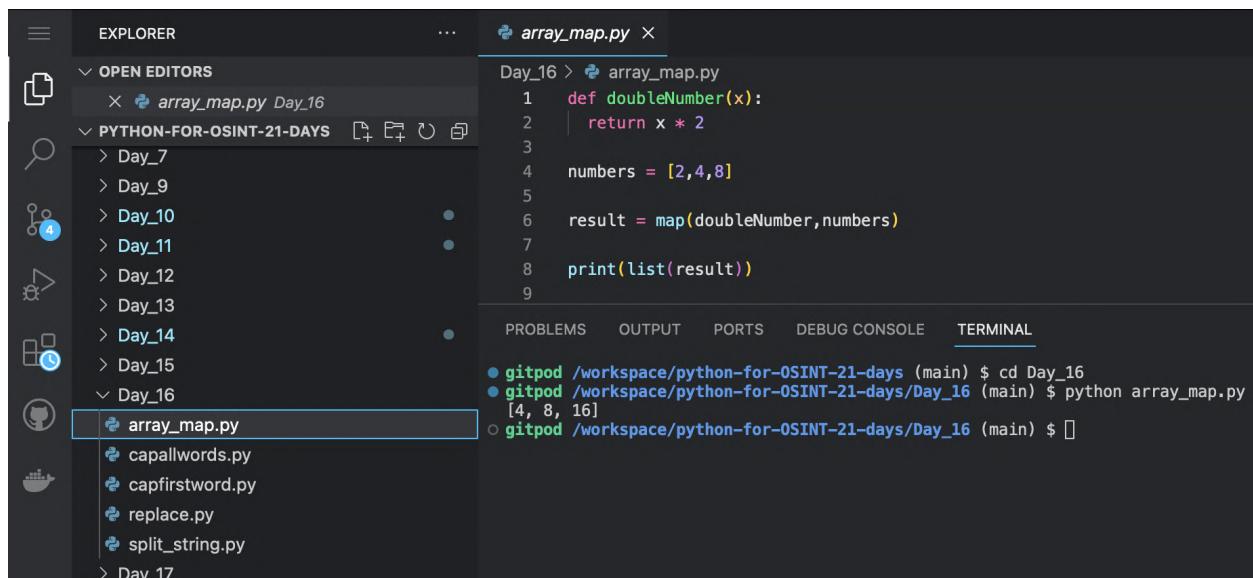
API-s-for-OSINT (Domain/DNS/IP lookup)

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:



```
array_map.py
Day_16 > array_map.py
1 def doubleNumber(x):
2 | return x * 2
3
4 numbers = [2,4,8]
5
6 result = map(doubleNumber,numbers)
7
8 print(list(result))
9
```

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL

```
gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_16
gitpod /workspace/python-for-OSINT-21-days/Day_16 (main) $ python array_map.py
[4, 8, 16]
gitpod /workspace/python-for-OSINT-21-days/Day_16 (main) $ 
```

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)
```

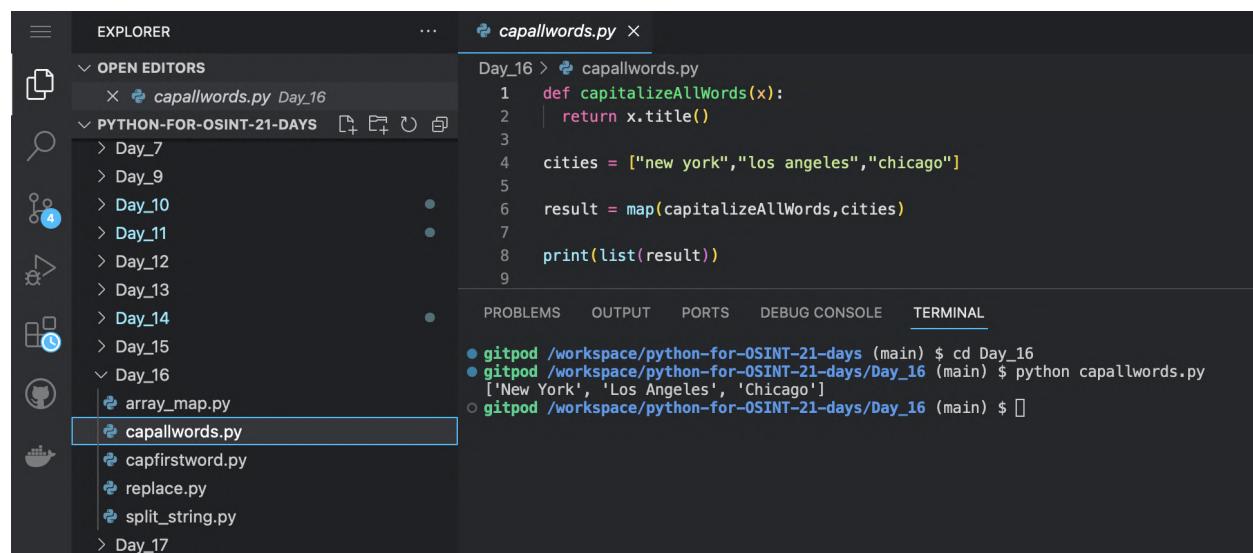
```
# Display the modified list:
```

```
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`:



```
EXPLORER ... capallwords.py ×  
OPEN EDITORS capallwords.py Day_16  
PYTHON-FOR-OSINT-21-DAYS Day_7 Day_9 Day_10 Day_11 Day_12 Day_13 Day_14 Day_15 Day_16 array_map.py capallwords.py capfirstword.py replace.py split_string.py Day_17  
PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL  
gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_16  
gitpod /workspace/python-for-OSINT-21-days/Day_16 (main) $ python capallwords.py  
['New York', 'Los Angeles', 'Chicago']  
gitpod /workspace/python-for-OSINT-21-days/Day_16 (main) $
```

```
# 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:

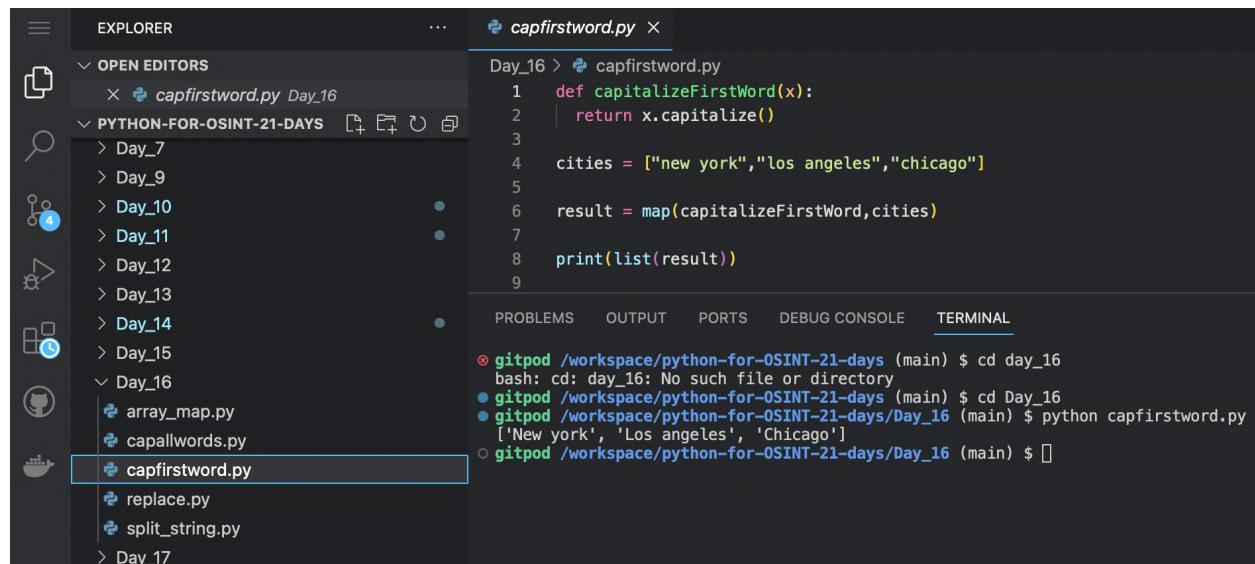
```
result = map(capitalizeAllWords,cities)
```

Display the modified list:

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

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

Run capfirstword.py:



The screenshot shows the Visual Studio Code interface. On the left is the Explorer sidebar, which lists several Python files in a folder named 'Day_16'. The file 'capfirstword.py' is currently selected. The main area is the code editor, displaying the following Python script:

```
def capitalizeFirstWord(x):  
    return x.capitalize()  
  
cities = ["new york","los angeles","chicago"]  
  
result = map(capitalizeFirstWord,cities)  
  
print(list(result))
```

Below the code editor is the Terminal panel, which shows the command-line output of running the script:

```
gitpod /workspace/python-for-OSINT-21-days (main) $ cd day_16  
bash: cd: day_16: No such file or directory  
gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_16  
gitpod /workspace/python-for-OSINT-21-days/Day_16 (main) $ python capfirstword.py  
['New York', 'Los Angeles', 'Chicago']  
gitpod /workspace/python-for-OSINT-21-days/Day_16 (main) $
```

```
# Create a function capitalizerFirstWord 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:

The screenshot shows the VS Code interface. The Explorer pane on the left lists files in a tree view. The terminal pane at the bottom shows a command-line session with several gitpod commands.

```
Day_16 > replace.py
1 def replaceDash(x):
2     return x.replace("_","-")
3
4 words = ["six_pack","king_size","editor_in_chief"]
5
6 result = map(replaceDash,words)
7
8 print(list(result))
9
```

● PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL

- gitpod /workspace/python-for-OSINT-21-days (main) \$ cd Day_16
- gitpod /workspace/python-for-OSINT-21-days/Day_16 (main) \$ python replace.py
- ['six-pack', 'king-size', 'editor-in-chief']
- gitpod /workspace/python-for-OSINT-21-days/Day_16 (main) \$

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"]
```

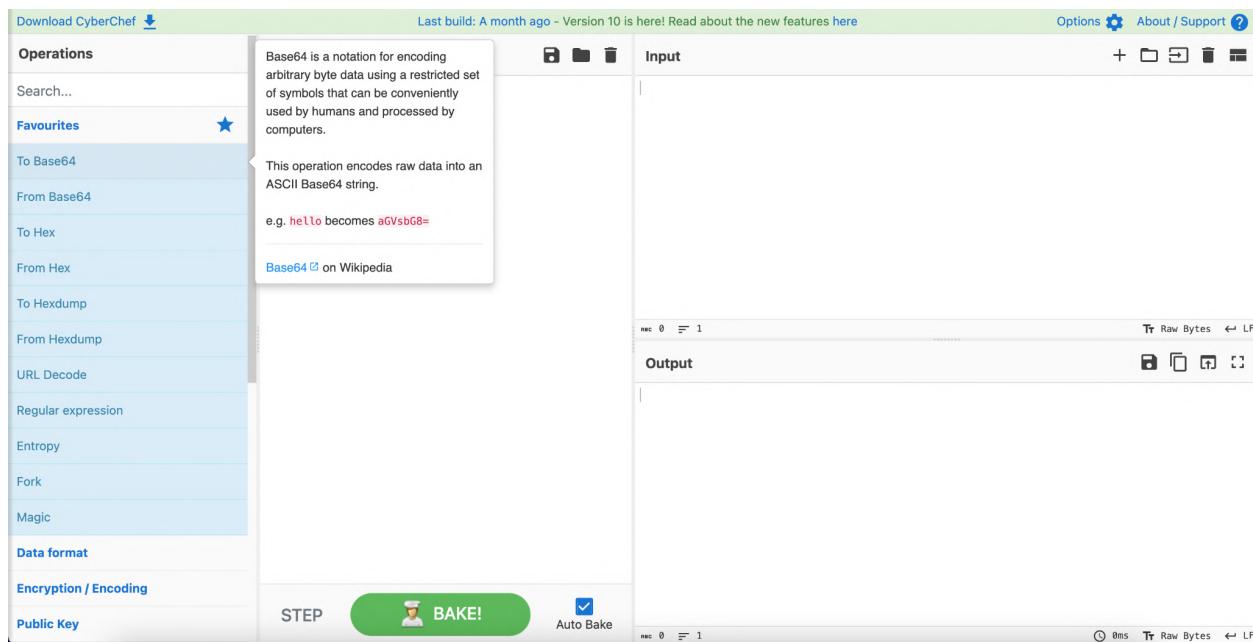
Run the map() function with arguments in the form of an array of three words and the replaceDash function:

```
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/cheipy/>). 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`:

The screenshot shows the VS Code interface. The Explorer sidebar on the left lists files in the workspace, including 'split_string.py Day.16' and several files under 'PYTHON-FOR-OSINT-21-DAYS' such as 'Day.7', 'Day.9', 'Day.10', etc. The Editor pane on the right displays the code for 'split_string.py':

```
Day_16 > split_string.py
1 string = "first name;last name@email;address;"
2
3 fields = string.split(";")
4
5 for field in fields:
6     print (field)
7
```

The Terminal pane at the bottom shows the execution of the script:

```
gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_16
gitpod /workspace/python-for-OSINT-21-days/Day_16 (main) $ python split_string.py
first name
last name
email
address
```

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

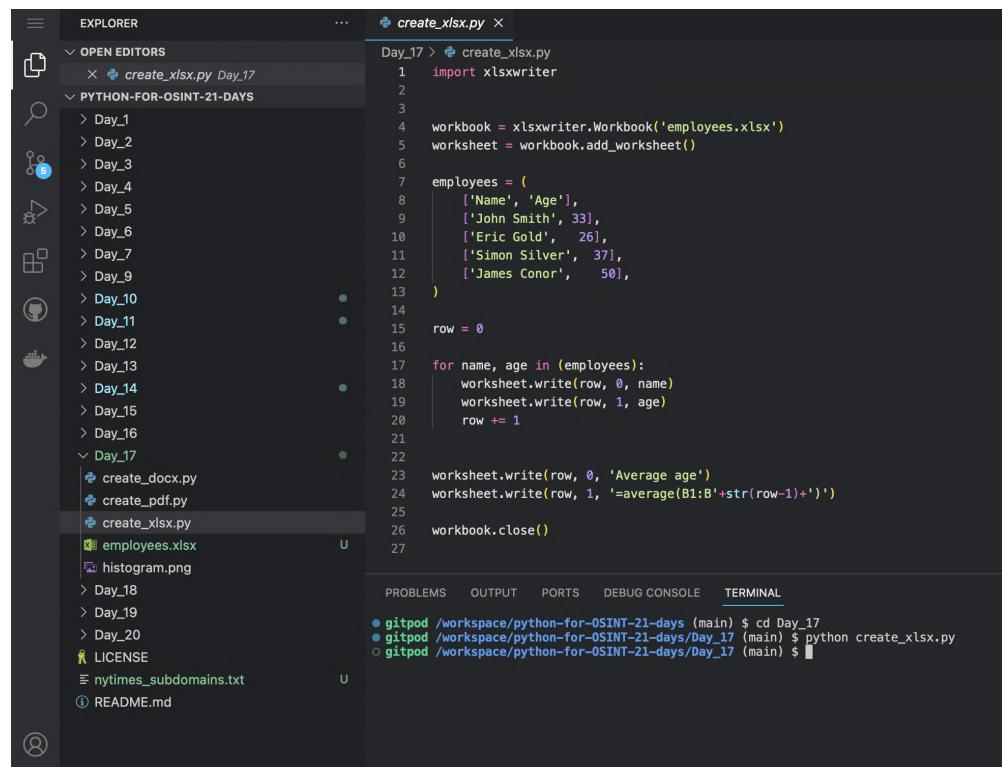
One important part of most research or investigation is the writing of the final report. In case you are creating many similar reports based on data with a similar structure, you can automate this process using Python. Let's try to create a simple MS Excel book.

```
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:



```
Day_17 > ⚡ create_xlsx.py
1 import xlsxwriter
2
3
4 workbook = xlsxwriter.Workbook('employees.xlsx')
5 worksheet = workbook.add_worksheet()
6
7 employees = (
8     ['Name', 'Age'],
9     ['John Smith', 33],
10    ['Eric Gold', 26],
11    ['Simon Silver', 37],
12    ['James Conor', 50],
13 )
14
15 row = 0
16
17 for name, age in employees:
18     worksheet.write(row, 0, name)
19     worksheet.write(row, 1, age)
20     row += 1
21
22 worksheet.write(row, 0, 'Average age')
23 worksheet.write(row, 1, '=average(B1:B'+str(row-1)+')')
24
25 workbook.close()
```

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

```
import xlsxwriter
```

Create a new file (book) named employees.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:

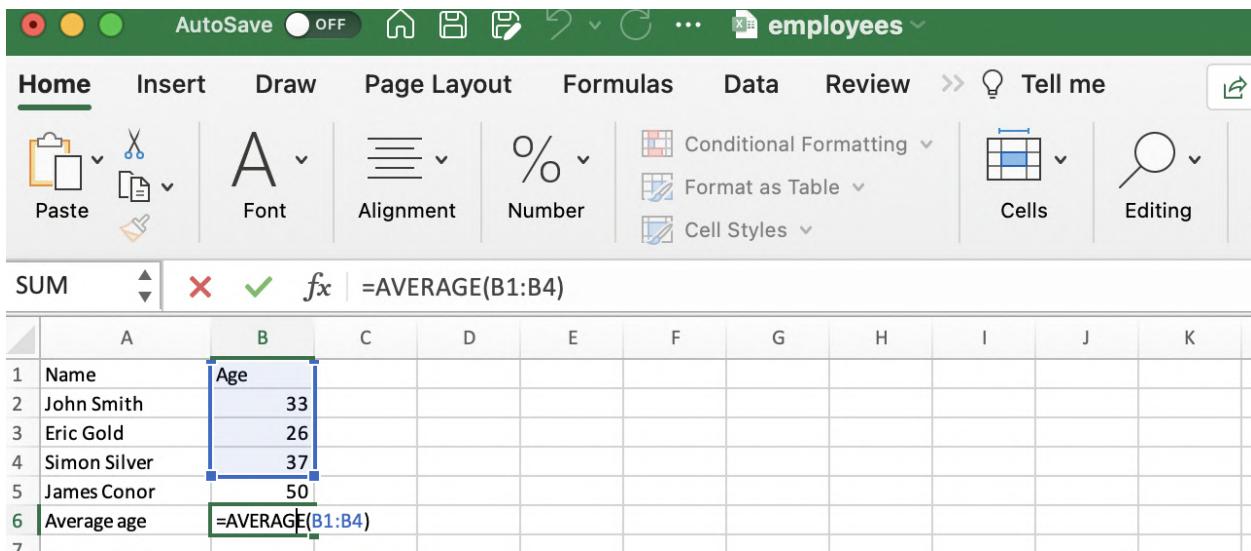
```
worksheet.write(row, 0, 'Average age')
```

```
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:



A screenshot of Microsoft Excel showing a table with data and a formula. The table has columns A through K. Row 1 contains "Name" and "Age". Rows 2, 3, 4, and 5 contain "John Smith", "Eric Gold", "Simon Silver", and "James Conor" respectively, with their ages 33, 26, 37, and 50 listed in column B. Row 6 contains "Average age" and the formula "=AVERAGE(B1:B4)" in column B. The formula bar at the top shows "SUM" and the formula "=AVERAGE(B1:B4)". The ribbon menu is visible at the top, showing tabs like Home, Insert, Draw, etc.

| | A | B | C | D | E | F | G | H | I | J | K |
|---|--------------|-----------------|---|---|---|---|---|---|---|---|---|
| 1 | Name | Age | | | | | | | | | |
| 2 | John Smith | 33 | | | | | | | | | |
| 3 | Eric Gold | 26 | | | | | | | | | |
| 4 | Simon Silver | 37 | | | | | | | | | |
| 5 | James Conor | 50 | | | | | | | | | |
| 6 | Average age | =AVERAGE(B1:B4) | | | | | | | | | |
| 7 | | | | | | | | | | | |

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=b7bc3f5e5dbc2e890532e6eaf57f7740702a4517dfc7036d862f07944a88856e
  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`:

The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left lists a folder named "PYTHON-FOR-OSINT-21-DAYS" containing subfolders "Day_1" through "Day_17". The current folder is "Day_17", which contains files like "create_docx.py", "create_pdf.py", "create_xlsx.py", "employees.xlsx", "histogram.png", "report.docx", and "LICENSE". The code editor on the right displays the content of "create_docx.py". The terminal at the bottom shows the command "python create_docx.py" being run, with the output "report.docx" generated.

```
Day_17 > create_docx.py
1  from docx import Document
2  from docx.shared import Inches
3
4  document = Document()
5
6  document.add_heading('Report', 0)
7
8  document.add_heading('Report', level=1)
9
10 document.add_paragraph(
11     'Some text in report'
12 )
13
14 document.add_page_break()
15
16 document.add_picture('histogram.png', width=Inches(1.25))
17
18 document.save('report.docx')
19
20
```

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL

```
● gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_17
● gitpod /workspace/python-for-OSINT-21-days/Day_17 (main) $ python create_docx.py
○ gitpod /workspace/python-for-OSINT-21-days/Day_17 (main) $ █
```

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()
```

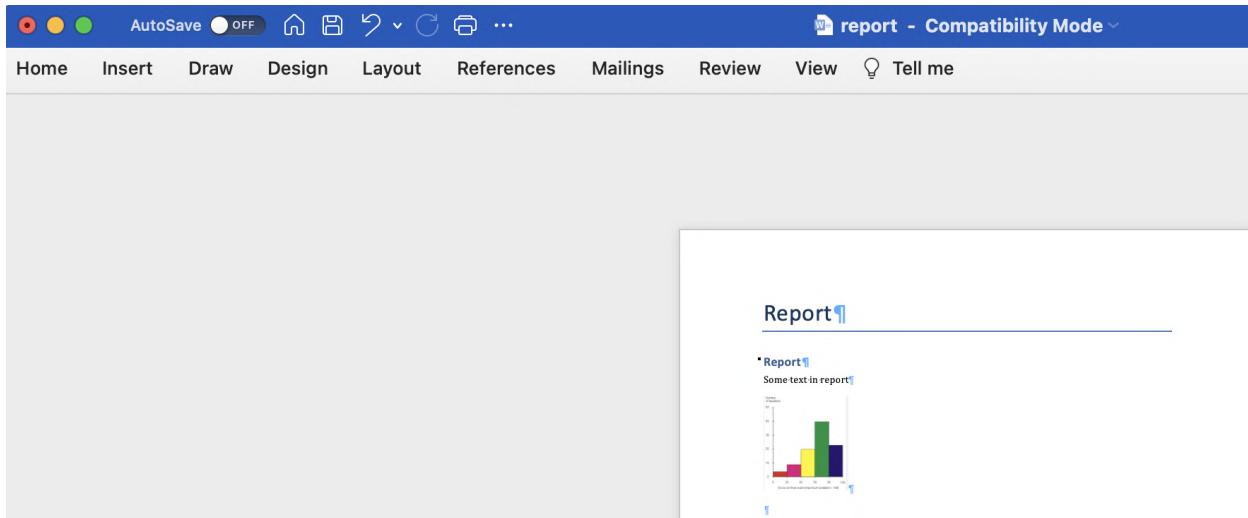
Add an image to the document:

```
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`:

```
Day_17 > create_pdf.py M X
1   from fpdf import FPDF
2
3
4   pdfFile = FPDF()
5
6   pdfFile.add_page()
7
8   pdfFile.set_font("Arial", size = 12)
9
10  pdfFile.cell(20, 10, txt = "Report text", ln = 2, align = 'C')
11
12  pdfFile.image('histogram.png', 10, 40, 30)
13
14  pdfFile.output("report.pdf")
15
```

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL

- gitpod /workspace/python-for-OSINT-21-days (main) \$ cd Day_17
- gitpod /workspace/python-for-OSINT-21-days/Day_17 (main) \$ python create_pdf.py
- gitpod /workspace/python-for-OSINT-21-days/Day_17 (main) \$ []

| File | Type |
|----------------|------|
| create_docx.py | M |
| create_pdf.py | M |
| create_xlsx.py | |
| employees.xlsx | U |
| histogram.png | |
| report.docx | U |
| report.pdf | U |

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')
```

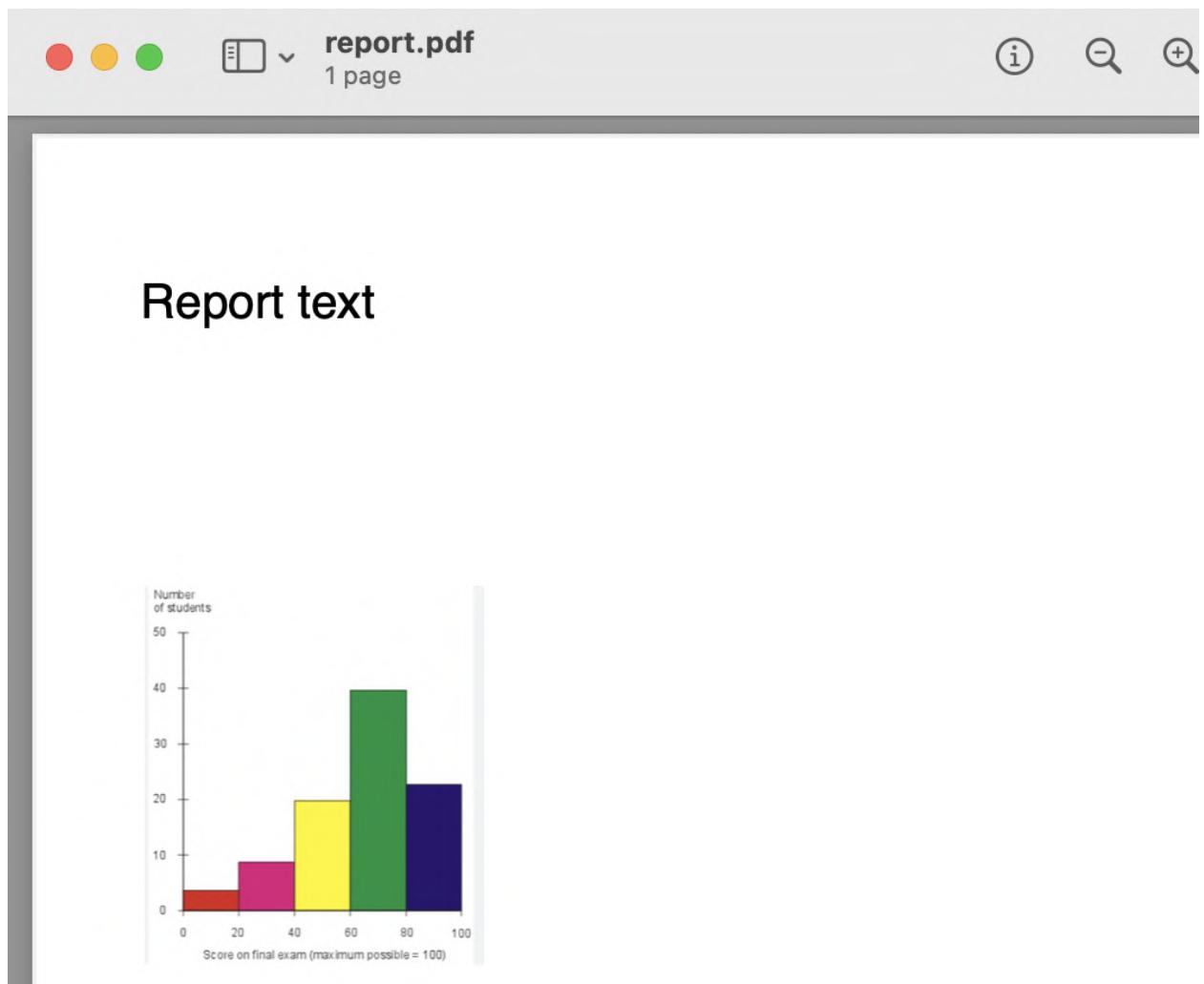
```
# Add an image to the page, specifying its size and the coordinates of the  
top and bottom indents:
```

```
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 matplotlib)
(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:

The screenshot shows the VS Code interface. The Explorer pane on the left displays a file tree for a workspace named 'PYTHON-FOR-OSINT-21-DAYS'. The tree includes subfolders for 'Day_1' through 'Day_18', and files like 'bar.py', 'map.py', 'population_barchart.png', 'LICENSE', 'nytimes_subdomains.txt', and 'README.md'. The 'Day_18' folder is expanded, showing its contents. The Terminal pane at the bottom has three entries:

- gitpod /workspace/python-for-OSINT-21-days (main) \$ cd Day_18
- gitpod /workspace/python-for-OSINT-21-days/Day_18 (main) \$ python bar.py
- gitpod /workspace/python-for-OSINT-21-days/Day_18 (main) \$

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 pypyproj<3.5.0,>=1.9.3
  Downloading pypyproj-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 matplotlib<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 matplotlib<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 matplotlib<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)
Requirement already satisfied: pillow<6.2.0,>=6.1.0 in /workspace/.pyenv_mirror/user/current/lib/python3.11/site-packages (from matplotlib<3.7,>=1.5->basemap) (6.2.0)
Requirement already satisfied: pytz in /usr/lib/python3.11/site-packages (from matplotlib<3.7,>=1.5->basemap) (2023.1)
Requirement already satisfied: six in /usr/lib/python3.11/site-packages (from matplotlib<3.7,>=1.5->basemap) (1.16.0)
Requirement already satisfied: shapely in /usr/lib/python3.11/site-packages (from matplotlib<3.7,>=1.5->basemap) (1.8.0)
Requirement already satisfied: wrapt in /usr/lib/python3.11/site-packages (from matplotlib<3.7,>=1.5->basemap) (1.14.1)

```

Install the basemap package before running script:

pip install basemap

Run map.py:

```

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
fig = plt.figure(figsize = (22,22))
map = Basemap()
map.drawcountries()
map.drawcoastlines()
map.plot(43.00, 39.00, 'bo', markersize=12)
plt.title("World map", fontsize=20)
plt.savefig('map.png')

```

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL

- gitpod /workspace/python-for-OSINT-21-days (main) \$ cd Day_18
- gitpod /workspace/python-for-OSINT-21-days/Day_18 (main) \$ python map.py
- gitpod /workspace/python-for-OSINT-21-days/Day_18 (main) \$

```
# Importing matplotlib, numpy, 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)
```

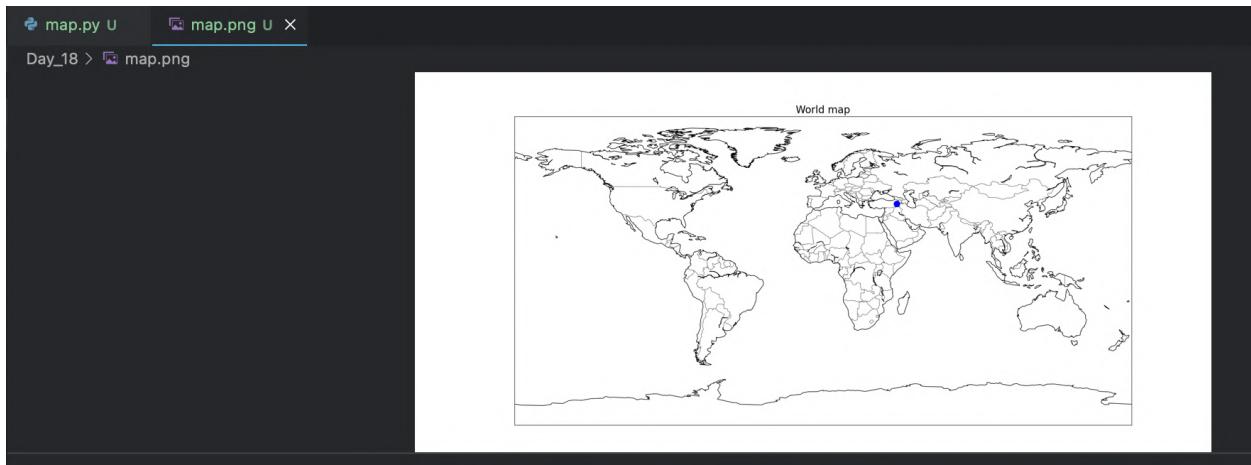
```
# Add title to the map:
```

```
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:



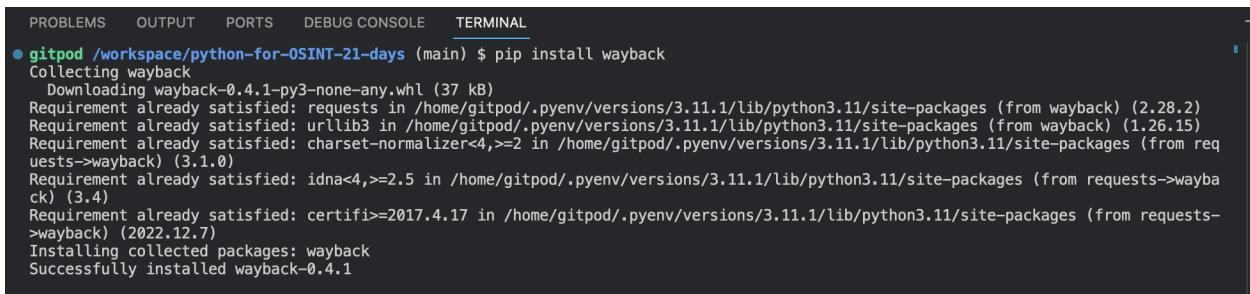
Basemap allows you to load any maps from shape files (with shp extension). This allows you to make visualizations for individual countries and regions. You can read more about it here:

<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.



```
PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL
● gitpod /workspace/python-for-OSINT-21-days (main) $ pip install wayback
Collecting wayback
  Downloading wayback-0.4.1-py3-none-any.whl (37 kB)
    Requirement already satisfied: requests in /home/gitpod/.pyenv/versions/3.11.1/lib/python3.11/site-packages (from wayback) (2.28.2)
    Requirement already satisfied: urllib3 in /home/gitpod/.pyenv/versions/3.11.1/lib/python3.11/site-packages (from wayback) (1.26.15)
    Requirement already satisfied: charset-normalizer<4,>=2 in /home/gitpod/.pyenv/versions/3.11.1/lib/python3.11/site-packages (from req
uests->wayback) (3.1.0)
    Requirement already satisfied: idna<4,>=2.5 in /home/gitpod/.pyenv/versions/3.11.1/lib/python3.11/site-packages (from requests-
>wayback) (3.4)
    Requirement already satisfied: certifi>=2017.4.17 in /home/gitpod/.pyenv/versions/3.11.1/lib/python3.11/site-packages (from requests-
>wayback) (2022.12.7)
  Installing collected packages: wayback
    Successfully installed wayback-0.4.1
```

Install wayback package before running script:

```
pip install wayback
```

Run download_mementos.py:

The screenshot shows the VS Code interface with the following details:

- EXPLORER** sidebar: Shows a tree view of files and folders. Under "OPEN EDITORS", there is a file named "download_mementos.py". Under "PYTHON-FOR-OSINT-21-DAYS", there are multiple "Day_X" folders (X from 1 to 20), a "date_time.py" file, and a "LICENSE" file.
- Code Editor**: The active editor contains the "download_mementos.py" script. The code uses the WaybackClient API to search for web pages from nasa.gov saved before January 1, 1999, and saves them as HTML files.
- TERMINAL**: The terminal shows the command "gitpod /workspace/python-for-OSINT-21-days (main) \$ cd Day_19" followed by the execution of the script. The output lists three URLs from the Wayback Machine corresponding to the specified date range.

```

import wayback
from datetime import datetime

client = wayback.WaybackClient()

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

    memento = client.get_memento(record)

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

    memento_file = open(fileName, "a")

    memento_file.write(memento.text)

    memento_file.close()

    print (fileName)

```

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`:

```
date_time.py
1 import datetime
2
3 currentTime = datetime.datetime.now()
4
5 print(currentTime)
6
7 print("Current Year: "+str(currentTime.year))
8
9 print("Current Month: "+str(currentTime.month))
10
11 print(currentTime.strftime("%A %d %B %Y"))
12
```

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL

- gitpod /workspace/python-for-OSINT-21-days (main) \$ cd Day_19
- gitpod /workspace/python-for-OSINT-21-days/Day_19 (main) \$ python date_time.py

2023-05-04 16:25:39.532865
Current Year: 2023
Current Month: 5
Thursday 04 May 2023

- gitpod /workspace/python-for-OSINT-21-days/Day_19 (main) \$

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 streamlit) (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
```

The screenshot shows a Gitpod interface with a terminal window open. The terminal output is as follows:

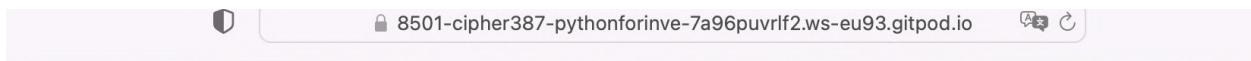
```
gitpod /workspace/python-for-OSINT-21-days (main) $ cd Day_20
gitpod /workspace/python-for-OSINT-21-days/Day_20 (main) $ streamlit run webapp.py
Collecting usage statistics. To deactivate, set browser.gatherUsageStats to False.

You can now view your Streamlit app in your browser.
Network URL: http://10.0.5.2:8501
External URL: http://34.79.95.243:8501
```

The terminal also lists several bash sessions for days 1 through 20.

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

If you use Gitpod, just click Open Browser.



Simple web app

Enter some text

yes

Start

You entered: yes

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.

The screenshot shows the ExtendsClass Python syntax checker interface. At the top, there's a navigation bar with 'About' and a link to 'Free Online Toolbox for developers'. Below the header, there are two buttons: 'Browse Python file' and 'Check Python syntax ▶'. The main area is divided into two sections: 'Python code' and 'Results'. In the 'Python code' section, there's a code editor containing the following Python script:

```
3 from mpl_toolkits.basemap import Basemap  
4  
5 fig = plt.figure(figsize = (22,22))  
6  
7 map = Basemap()  
8  
9 map.drawcountries()  
10 map.drawcoastlines()  
11  
12 map.plot(43.00, 39.00, 'bo', markersize=12)  
13  
14 plt.title("World map", fontsize=20)  
15  
16 plt.savefig('map.png')
```

In the 'Results' section, a pink box displays the error message: "Syntax errors detected :". It highlights line 5 with the text "fig = plt.figure(figsize = (22,22))" and an upward arrow, followed by the error message "SyntaxError: invalid syntax".

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

PythonChecker Makes Your Code Great Again

88%

Excellent

Lines: 16
Hints: 2

Your Code Sucks?



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

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

map = Basemap()

map.drawcountries()
map.drawcoastlines()

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

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

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

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.

This code is using three libraries - NumPy, Matplotlib and Basemap - to create a world map and plot a blue dot on the coordinates (43.00, 39.00).

First, the code imports the required libraries using the `import` statement. `numpy` is imported as `np` for convenience, `matplotlib.pyplot` is imported as `plt`, and `Basemap` is imported from the `mpl_toolkits.basemap` library.

Next, a new `figure` object is created using the `plt.figure()` method, with a size of 22x22 inches.

A new `Basemap` object is created using the default constructor. This creates a new map with default settings, centered at (0, 0) and spanning the entire globe.

The `drawcountries()` and `drawcoastlines()` methods are called on the `Basemap` object to draw the country borders and coastlines on the map.

Finally, a blue dot is plotted on the map using the `plot()` method with the coordinates (43.00, 39.00), and saved as a PNG file using the `savefig()` method with the filename 'map.png'.

The code ends with a title added to the plot using the `title()` method with the text "World map" and a font size of 20.

[Regenerate response](#)

Output in English Tone Default Writing Style Default Continue

ChatGPT Mar 23 Version. Free Research Preview. ChatGPT may produce inaccurate information about people, places, or facts.

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.

The screenshot shows the Stack Overflow homepage with a sidebar on the left. The sidebar has sections for Home, PUBLIC (Questions, Tags, Users, Companies), COLLECTIVES (Explore Collectives), and TEAMS (Stack Overflow for Teams). The main content area displays a search bar with '[python]', a title 'Questions tagged [python]', a 'Ask Question' button, and a note about Python's multi-paradigm nature. Below this is a summary of 2,126,413 questions with filters for Newest, Active, Bountied (30), Unanswered, and More. A specific question is highlighted: 'How can I change a column to a column from another dataframe based on a condition?' by Andraws Santos, posted 6 mins ago.

[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

New Thread

SESSION #1

```
what this code to do. import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap

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

map = Basemap()

map.drawcountries()
map.drawcoastlines()

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

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

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

319/3000

Expert Concise Creative

what this code to do. import numpy as np import matplotlib.pyplot as plt from mpl_toolkits.basemap import Basemap fig = plt.figure(figsize = (22,22)) map = Basemap() map.drawcountries() map.drawcoastlines() map.plot(43.00, 39.00, 'bo', markersize=12) plt.title("World map", fontsize=20) plt.savefig('map.png')

ANSWER

The code provided imports three libraries: numpy, matplotlib.pyplot, and Basemap. It then creates a new figure with a size of 22x22 using `fig = plt.figure(figsize = (22,22))`. Next, it creates a new Basemap object using `map = Basemap()`. The Basemap object is used to draw country borders and coastlines using `map.drawcountries()` and `map.drawcoastlines()`. Finally, the code plots a blue dot on the map at the coordinates (43.00, 39.00) using `map.plot(43.00, 39.00, 'bo', markersize=12)` and saves the resulting figure as a PNG file named 'map.png' using `plt.savefig('map.png')`. matplotlib.org

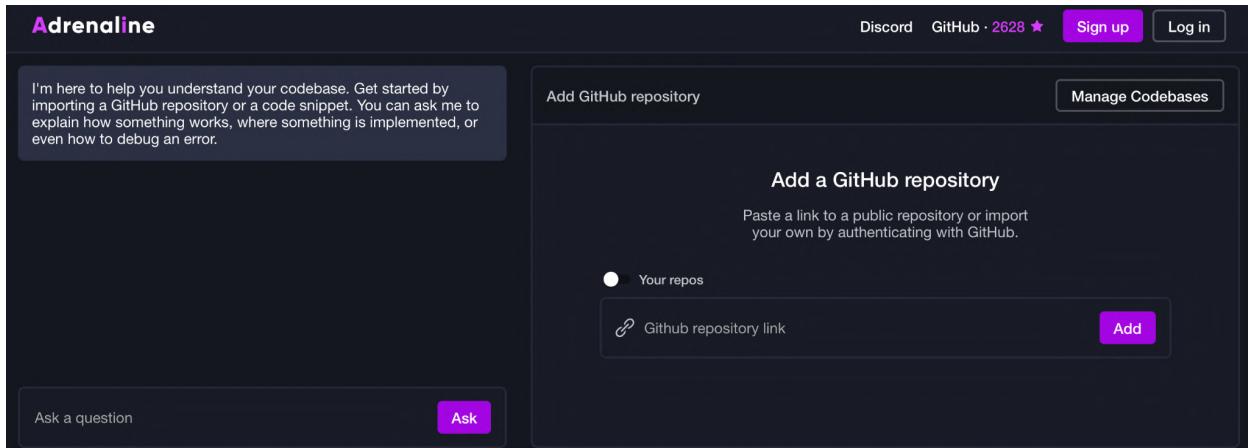
The `plot()` function is a method in the `matplotlib.pyplot` module that is used to draw lines and/or markers on a graph. It takes two arguments, `x` and `y`, which are the data to be plotted on the x and y axes, respectively. The `plot()` function can also take additional parameters to customize the appearance of the plot, such as the line color, line style, and marker style. In the provided code, the `plot()` function is used to plot a blue circle marker at the coordinates (43.00, 39.00) on the map. matplotlib.org

Basemap is a matplotlib toolkit for plotting 2D data on maps in Python. It is used to create maps with different projections, such as cylindrical or conic projections, and to add features to the map such as coastlines, country borders, and geographic labels. Basemap is no longer actively developed, and users are encouraged to switch to Cartopy, a similar package that is actively maintained. matplotlib.org

There are several approaches to installing matplotlib. One way is to use pip, a package installer for Python, to install matplotlib. To install matplotlib using pip, open a terminal or command prompt and type `pip install matplotlib`. If you are using a virtual environment, be sure to activate it before running the `pip install` command. Another approach is to install matplotlib using a package manager such as Anaconda or Homebrew. [\[2\]](#) [\[4\]](#)

If you encounter an error such as "No module named 'matplotlib'", it may be because matplotlib is not installed or is not installed correctly. In this case, you can try reinstalling matplotlib using pip or another package manager. If you are using a Jupyter notebook, you may need to install matplotlib and import it before importing other libraries such as pandas. You can do this by running the command `import matplotlib.pyplot as plt; %matplotlib inline` before importing pandas. [\[2\]](#) [\[4\]](#)

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

| | |
|--|-----|
| Who Is This Course For? | 2 |
| Who should avoid this course? | 3 |
| How to take this course | 4 |
| Day 0. Preparing for work | 6 |
| Day 1. Run the first script | 9 |
| Day 2. Minimum Basic Syntax | 13 |
| Day 3. Install and run Python command line tools | 20 |
| Day 4. Reading and writing files | 26 |
| Day 5. Handling HTTP requests and working with APIs | 33 |
| Day 6. JSON | 38 |
| Day 7. CSV | 43 |
| Day 8. Databases | 52 |
| Day 9. Automate the collection of search results | 56 |
| Day 10. Scraping | 64 |
| Day 11. Regular expressions | 69 |
| Day 12. Proxies | 72 |
| Day 13. Functions for working with lists | 79 |
| Day 14. Working with the file system | 85 |
| Day 15. Domain information gathering | 91 |
| Day 16. List mapping and functions for work with strings | 99 |
| Day 17. Generating documents | 106 |
| Day 18. Generating charts and maps | 115 |
| Day 19. Wayback Machine and time/date functions | 121 |
| Day 20. Web apps creation | 126 |
| Day 21. Tools to help you work with code | 130 |
| What to do next | 136 |
| Table of contents | 137 |