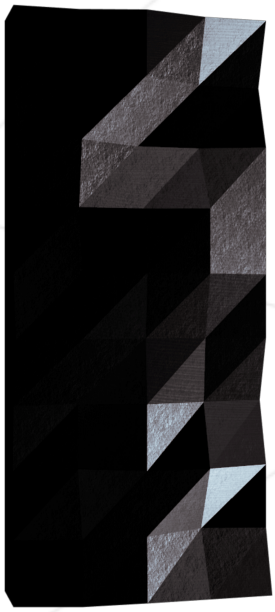

Module 00 – Piscine Python for Data Science

UNIX Command Line Tools

Summary: On the first day, we will help you to acquire the skills of using UNIX command-line tools for basic data science tasks. You will learn how to use curl, sort, uniq, jq, sed, and cat for data collection and preprocessing.

Contents

Chapter I

Foreword

We as Humanity has long known that data helps us to make better decisions. In Ancient Egypt, the government would conduct censuses to get a better understanding of how much taxes they could gather from the population. Even earlier, shepherds would count livestock to find out how many animals they could sell and how many they needed for the production of goods.

Since then, we have been developing more and more sophisticated algorithms for data processing. Now, we are able to replace something that we do not know with a prediction from machine learning algorithms. This helps us to prepare for the future: to predict demand for our goods and to make the according adjustments in our facilities. We can predict whether a person will return their credit or not so we can put our money aside for others and reap greater profits.

We have not only been developing algorithms but technologies and tools that have made data analysis cheaper and more convenient. They have democratized the whole field of data. Todays, it is much easier for a company to start using data for its own benefit. That is why there is so much hype around big data, artificial intelligence, and other such buzzwords.

Everybody can use data. Everybody can get value from it. Not only those who have a lot of money and resources, as was the case in the past.

As noted in the TV series Mr. Robot, “it’s an exciting time in the world right now”.

Chapter II

Instructions

- Use this page as your only reference. Do not listen to any rumors and speculations about how to prepare your solution.
- Here and further on we use Python 3 as the only correct version of Python.
- The python files for python exercises (module01, module02, module03) must have the following block at the end: if __name__ == '__main__'.
- Pay attention to the permissions of your files and directories.
- To be assessed your solution must be in your GIT repository.
- Your solutions will be evaluated by your piscine peers.
- You should not leave any other files in your directory other than those explicitly specified in the exercise instructions. It is recommended that you modify your .gitignore to avoid any accidents.
- When you need to get precise output in your programs, it is forbidden to display a precalculated output instead of performing the exercise correctly.
- Have a question? Ask your neighbor on the right. If that fails, try your neighbor on the left.
- Your reference materials: peers / Internet / Google.
- Remember to engage in discussion on the Intra Piscine forums.
- Read the examples carefully. They may require things that are not otherwise specified in the subject.
- And may the Force be with you!

Chapter III

Exercise 00 : First shell script

	Exercise 00
First shell script	
Turn-in directory : *ex*00/	
Files to turn in : hh.sh, hh.json	
Allowed functions : curl, jq	

In this exercise, you will need to interact with the HeadHunter API to parse some information about vacancies. In order to do this, you will need to understand how both curl and the HeadHunter API work.

Write a shell script that:

- gets the name of a vacancy - 'data scientist' as an argument (some later exercises will be based on this),
- downloads information about the first 20 vacancies corresponding to the search parameters,
- stores it in a file with the name hh.json.

The result in the file must be formatted in such a way that each field is placed on a different line. See the example below:

```
{
  "page": 0,
  "found": 344,
  "clusters": null,
  "arguments": null,
  "per_page": 20,
  "pages": 18,
  "items": [
    {
      "apply_alternate_url": "https://hh.ru/applicant/vacancy_response?vacancyId=35895583",
      "address": {
        "id": "118668",
        "lat": 55.762556,
        "metro": {
          "station_id": "7.67",
          "line_name": "Таганско-Краснопресненская",
          "lng": 37.624423,
          "line_id": "7",
          "station_name": "Кузнецкий мост",
          "lat": 55.761498
        },
        "street": "Кузнецкий мост",
        "lng": 37.627175,
        "metro_stations": [
          {
            "line_name": "Таганско-Краснопресненская",
            "station_id": "7.67",
            "lng": 37.624423,
            "line_id": "7",
            "lat": 55.761498,
            "station_name": "Кузнецкий мост"
          }
        ],
        "building": "21/5",
        "city": "Москва",
        "description": null,
        "raw": "Москва, Кузнецкий мост, 21/5"
      }
    }
  ]
}
```


Your script must be executable. The interpreter to use is /bin/sh.

Put your script as well as your result of parsing in the folder ex00 in the root directory of your repository.

Chapter IV

Exercise 01 : Transforming JSON to CSV

	Exercise 01
Transforming JSON to CSV	
Turn-in directory : *ex*01/	
Files to turn in : filter.jq, json_to_csv.sh, hh.csv	
Allowed functions : jq	

What you got in the previous exercise was a JSON file. It is a popular file format for APIs but can be inconvenient for actual data analysis. So, you will need to convert it into a more convenient CSV file.

Write a shell script called json_to_csv.sh that:

- executes jq with a filter written in a separate file filter.jq
- filters the following 5 columns corresponding to the vacancies: “id”, “created_at”, “name”, “has_test”, and “alternate_url”
- saves the result to the CSV file hh.csv

See the example below:

```
“id”,”created_at”,”name”,”has_test”,”alternate_url”
"35895583","2020-04-12T12:06:33+0300","Специалист / data scientist (big data, прогностическая аналитика,
 data mining)",false,"https://hh.ru/vacancy/35895583"
"36359628","2020-04-11T19:25:48+0300","Senior Data Scientist",false,"https://hh.ru/vacancy/36359628"
"35218725","2020-04-11T18:03:53+0300","Junior Data scientist",false,"https://hh.ru/vacancy/35218725"
```

The CSV file must have headers in the first row.

Your script must be executable. The interpreter to use is /bin/sh.

Put your filter file - the file that converts JSON to CSV, as well as the result of your conversion in the ex01 folder in the root directory of your repository.

Chapter V

Exercise 02 : Sorting a file

	Exercise 02
Sorting a file	
Turn-in directory : *ex*02/	
Files to turn in : sorter.sh, hh_sorted.csv	
Allowed functions : cat, sort, head, tail	

Sometimes having your data in a non-random order, having it but sorted in some way can be efficient for later stages of data analysis. So in this exercise, you will need to sort your CSV file with several columns.

Write a shell script called sorter.sh that:

- sorts the hh.csv file from the previous exercise according to the column "created_at" and then by the "id" in ascending order
- saves the result in the CSV file hh_sorted.csv

The CSV file must still have headers in its first row.

Your script must be executable. The interpreter to use is /bin/sh.

Put your shell script as well as your result of the sorting in the folder ex02 in the root directory of your repository.

Chapter VI

Exercise 03 : Replacing strings in a file

	Exercise 03
Replacing strings in a file	
Turn-in directory : $ex03/$	
Files to turn in : cleaner.sh, hh_positions.csv	
Allowed functions : n/a	

Raw data is a mess. Before you can start analyzing it, you need to do a lot of preprocessing. In this exercise, that is what you are continuing to do. If you look at your file from the previous exercise, you will see that every position name of contains “Data Scientist”(you don’t have to check this). This is not surprise since we used that string as the keyword for the search in the HeadHunter API. But for us, as for the algorithms, it does not give any useful information. To be honest, it is just noise that worsens data analysis.

Write a shell called script cleaner.sh that:

- takes “Junior”, “Middle”, “Senior” from the names of position, if the name does not contain any of these words use “-” (e.g. “Senior Data Scientist” -> “Senior”, “analyst /(data scientist)” -> “-”, “Специалист / data scientist (big data, прогностическая аналитика, data mining)” -> “-”), if there are several of them, keep them all(e.g. “Middle/Senior Data Scientist” -> “Middle/Senior”)
- saves the result in the CSV file hh_positions.csv.

You can see the example below:

```
"id","created_at","name","has_test","alternate_url"
"35218725","2020-04-11T18:03:53+0300","Junior",false,"https://hh.ru/vacancy/35218725"
"36359628","2020-04-11T19:25:48+0300","Senior",false,"https://hh.ru/vacancy/36359628"
"35895583","2020-04-12T12:06:33+0300","-",false,"https://hh.ru/vacancy/35895583"
```

The CSV file must still have headers in its first row and be sorted as per the previous exercise.

Your script must be executable. The interpreter to use is /bin/sh.

Put your shell script as well as your result from cleaning in the ex03 folder in the root directory of your repository.

Chapter VII

Exercise 04 : Descriptive statistics

	Exercise 04
Descriptive statistics	
Turn-in directory : *ex*04/	
Files to turn in : counter.sh, hh_uniq_positions.csv	
Allowed functions : n/a	

Before doing anything more sophisticated, it is best to get a basic knowledge of your data. In this exercise, you will need to count the unique positions in your file. As a result, you can understand that your data skewed somehow: for instance, there are more seniors than juniors. Such facts might be useful for further analysis.

Write a shell script called counter.sh that:

- counts unique values of the name column in the file you prepared in the
- previous exercise,
- sorts the table by that count in descending order
- stores the result in the CSV file hh_uniq_positions.csv

See the example below:

```
"name","count"
"Junior",10
"Middle",5
"Senior",3
```

The CSV file must have headers in the first row as in the example.

Your script must be executable. The interpreter to use is /bin/sh.

Put your shell script as well as the result of counting in the ex04 folder in the root directory of your repository.

Chapter VIII

Exercise 05 : Partitioning and concatenation

	Exercise 05
Partitioning and concatenation	
Turn-in directory : $ex05/$	
Files to turn in : partitioner.sh, concatenator.sh	
Allowed functions : n/a	

When you have a big dataset, sometimes it might be useful to slice it into partitions. Each partition has a specific range of keys. One of the popular ways to partition is to do it by date. Each partition contains data on a specific date. In this exercise, you will need to perform that task.

Write one shell script called partitioner.sh that:

- takes as input the result of Exercise 03
- stores slices of data with different "created_at" dates in separate CSV files named for that date

See the example of such a file below:

```
"id","created\_at","name","has\_test","alternate\_url"
"35218725","2020-04-11T18:03:53+0300","Junior",false,"https://hh.ru/vacancy/35218725"
"36359628","2020-04-11T19:25:48+0300","Senior",false,"https://hh.ru/vacancy/36359628"
```

Write another shell script called concatenator.sh that:

- takes as input the separate files from the result of partitioner.sh
- concatenates all separate files into one CSV file

The CSV files must have headers in the first row, as in the example. The CSV from the result of concatenator.sh must be equal to the result of Exercise 03.

Your scripts must be executable. The interpreter to use is /bin/sh.

Put your shell scripts in the ex05 folder in the root directory of your repository.