**Speechmatics Python Coding Exercise**

Contact: [careers@speechmatics.com](mailto:careers@speechmatics.com)

# Introduction

The Speechmatics Python Coding Exercise is designed to test your ability to write Python code by presenting you with a set of requirements for an independent Python application. We are looking for you to demonstrate that you can accurately interpret the requirements and translate them into high quality code that works and is be ready to be used in a production environment.

Typically, we will ask you to complete the exercise independently and provide us with your solution. You should then provide us with either a tarball / zip containing your solution or a link to a github repo we can download a solution from.

# What we are looking for

1. **Fulfils requirements**. Our preference is for working code that fulfils the requirements over half-finished code that may have looked or performed better.
2. **Readability**. Code should be easy to follow and understand, with comments where appropriate (you may assume the reader is proficient in Python).
3. **Robustness**. Your code should not catastrophically fail under reasonable use.
4. **Performance**. Your code will be measured for speed and memory usage.
5. **Timely**. We expect this candidate to take most candidates 2-3 hours. Please do not spend longer than 4 hours on it – if you reach this limit please just include whatever you have produced so far. Please indicate approximately how long you spent on your solution in your response.
6. **Pragmatism**. The requirements are not exhaustive and there may be areas where you do not fully understand them, or need to make a compromise due to a constraint (e.g., time pressure). We are looking for you to demonstrate you can make pragmatic decisions about details here.
7. **Advanced Python**. For top marks you will need to demonstrate to us that you understand and can use Python best practice, including, for example, using logging.

# Background

Imagine that we store our static data files on a single isolated server behind a RESTful API. Whenever we want to use one of those data files we make an API request to the server and download the file for use.

The data is organised by language, such that a request will be to a URL that can be broken down like this:

**https://speechmatics.data.com/English/some\_audio\_file.wav**

<https://speechmatics.data.com> – this is our base server url (this part will be omitted in logs)

English – this is the language the data is being drawn from

some\_audio\_file.wav – this is the actual data file name, which could be anything

We want to monitor our usage of this data server so that we can determine if we can optimise our practices. To help do this we store access logs for the server in the Apache Common Log Format. Your task is to design a Python program that can parse these logs and produce a standardised report on monthly usage of certain aspects of the server.

# Requirements

## Your Code

1. Your code should be written in Python 3. Please indicate which exact version of Python you have used for development and testing.
2. Your solution will be tested in Ubuntu 16.04. However, it should be suitable to be run on a range other OSs.
3. Your code should contain one file called ‘log\_parser.py’ and run in the following way:  
   `python3 log\_parser.py $directory\_path $output\_path`  
   where $directory\_path is an absolute *or* relative path to a directory containing log files  
   $output\_path is an absolute *or* relative path to an output file that will be created or overwritten with the output of your program
4. You may (and should!) use any standard Python library, but not third-party libraries.
5. Your code should be original and not copied from any other source. If you do believe there is a good reason to use external code, please indicate where and why you have done this clearly in your solution.

## Your Solution

Your solution should process a set of access log files and produce a report in JSON format, containing one section per month containing three types of statistic:

1. **A sorted list of the top 5 languages, ranked by total GB of data served from requests for files of that language for that month. The output should record total GB served for each language alongside the mean (average) and standard deviation request size in MB**
2. **The percentage of requests that were successful (2xx return code) that month**
3. **A list of all requested filenames that contained non-Ascii characters that month**

The script should take a directory as the first command line argument and assume all files within that directory are log files (you may assume there are no subdirectories). The Apache Common Log Format is used in these log files.

Your output should be saved to a file specified in the second command line argument and should be in a JSON format that exactly follows the following example.

The contents of the JSON are a list, where each item in the list represents a single month and is comprised of a dictionary of data. The months should be in date order, oldest first.

You may assume that users of the script will be computer literate, but not necessarily proficient in Python, so output such as stack backtraces should be avoided.

You should have been provided with a file named ‘example.log’ that you can use to test your code with.

## Clarifications

* For calculating bytes of data accessed per language, only successful requests (2xx responses) should be considered
* All requests may be treated the same regardless of method (e.g., GET, POST, PUT, etc.)
* You may assume all log lines report from the same time zone, and that time zone is the one you need to produce monthly reports for
* When reporting non-ascii filenames, report all of them that were requested whether the request was successful or not

## Example Required Output in JSON format:

[

{

"languages": [

{

"mean\_MB": 274.67621264705883,

"name": "English",

"stddev\_MB": 23.87456854020302,

"total\_GB": 23.347478075

},

{

"mean\_MB": 262.99960329411766,

"name": "Latin",

"stddev\_MB": 32.70786468839807,

"total\_GB": 13.412979768

},

{

"mean\_MB": 271.6549358181818,

"name": "Korean",

"stddev\_MB": 33.72303476254484,

"total\_GB": 11.952817176

},

{

"mean\_MB": 354.0662197878788,

"name": "Burmese",

"stddev\_MB": 47.80144861920779,

"total\_GB": 11.684185253

},

{

"mean\_MB": 314.8712610833333,

"name": "Minangkabou",

"stddev\_MB": 47.10816719253855,

"total\_GB": 11.335365399

}

],

"month": "December",

"non\_ascii": [],

"requests": {

"percent\_success": 75.2411575562701,

"success": 468,

"total": 622

},

"year": "2017"

},

{

"languages": [

{

"mean\_MB": 328.5391417619048,

"name": "English",

"stddev\_MB": 32.472809254535726,

"total\_GB": 20.697965931

},

{

"mean\_MB": 272.90769478723405,

"name": "Minangkabou",

"stddev\_MB": 39.69375134607011,

"total\_GB": 12.826661655

},

{

"mean\_MB": 286.354472175,

"name": "German",

"stddev\_MB": 38.72072679181873,

"total\_GB": 11.454178887

},

{

"mean\_MB": 292.45601464102566,

"name": "Korean",

"stddev\_MB": 45.95866777675219,

"total\_GB": 11.405784571

},

{

"mean\_MB": 373.120367,

"name": "Latin",

"stddev\_MB": 56.03487775585108,

"total\_GB": 11.19361101

}

],

"month": "January",

"non\_ascii": [],

"requests": {

"percent\_success": 74.11003236245955,

"success": 458,

"total": 618

},

"year": "2018"

}

]

The above example is an excerpt of two months from what we would expect from your code to report on the data provided.

## Apache Common Log Format

The format of the access logs is a space-separated list of fields, some of which are enclosed in double quotes. In order the fields are:

* **IP Address**  - the IP address of the client machine making the request
* **RFC-1413 ID -** the identity of the remote user (can be assumed to always be ‘-‘ in these logs)
* **User ID –** the logged in user (can be assumed to always be ‘-‘ in these logs)
* **Time –** The time of the request in the form [DD/MM/YYY:hh:mm:ss +ZZZZ] where ZZZZ is the time zone specification in a four digit offset from UTC.
* **Request –** The request method, URL and HTTP version enclosed in quotes (e.g., “GET /path HTTP/1.1”). (The URL can be assumed to always be of the form ‘/{language}/{filename}’ in these logs)
* **Status –** The HTTP status code of the response as an integer (e.g., 200)
* **Bytes –** The total number of bytes served in the response unit

If a double-quote character occurs in a double-quoted field, it is escaped by prefixing it with a backslash. Similarly, backslash characters themselves are represented by a double-backslash.

A sample log line in this format is shown below (note that it is wrapped for space reasons here, but it represents a single line in the real log file):

127.0.0.1 - - [01/01/2000:00:00:00 +0000] “GET /English/some\_audio\_file.wav HTTP/1.1” 200 1234567