UNIVERSITY OF SKÖVDE
SCHOOL OF INFORMATICS

SCRIPT PROGRAMMING 7.5 HP
IT384G

Spring Term 2025

ASSIGNMENT 1
PYTHON

Thomas Fischer ⟨thomas.fischer@his.se⟩

Aim

The lab assignment aims to provide the student skills in:

- Use regular expressions to match patterns
- Systematically design, implement, and troubleshoot scripts based on problems related to system administration

Contents

Please read this document carefully in its entirety before starting to program. If you have questions, ask a supervisor during the scheduled lab session or post your question in the course's forum in Canvas.

This assignment will be supervised by:

- Thomas Fischer
- Erik Andersson, Calle Dargren, and Ludvig Wiesner (student supervisors)

1 EXAMINATION

The Python lab assignment examines Assignment 1 (*Inlämningsuppgift 1*) of this course.

This assignment is an *individual assignment* and is graded according to the ECTS scale (A–E, and F). Grades A–E are passing grades.

Please read the text on plagiarism on the course's webpage. By submitting your code via Canvas or demonstrating your solution you acknowledge that you have read and understood this text on plagiarism.
Being an individual examination, you cannot use someone else's code, such as from your peers (past or present), the Internet, ChatGPT, or other sources.

1.1 Grade Settings

In order to achieve the lowest passing grade for this examination (grade E), you must:

- Complete all mandatory tasks of the assignment within the deadlines stated on the course page.
- Demonstrate and explain your solutions to one of the supervisors during one of the scheduled lab sessions.

We place great emphasis on your ability to be able to explain the code you have written in a detailed manner without any aids such as in the form of source code comments, written documentation, lab notes, or nearby students. For Python demonstrations, a helper script is provided that visualizes your code without comments without removing the comments from the original source code files.

- Submit your source code and `users.csv` as-is in one `.zip`, `.tar.gz`, or `.tar.xz` file on Canvas omitting unnecessary files.

You can use the following Bash code to pack the files:

```
tar -zcvf /tmp/python-code.tar.gz -C ~npc/Python merge-grades.py
  new-account-names.py small-game.py users.csv apachelog.py
  apachelog-test.py
```

Exclude temporary files or directories, Word documents, hidden files or directories, `__pycache__`, and `.vscode`. Do not include `access.log` or `access-small.log`.

To achieve grades D to A on this assignment, beyond fulfilling all requirements for grade E, you must earn 'extra points'. Each of the four available extra points improves your grade for this assignment by one level. For example, with two extra points your grade for this assignment will be grade C. Extra points can be earned by completing any of the optional parts of the four tasks (one extra point per task).

The 'grade' you receive for the demonstration you have to do during the lab sessions is only for the demonstration itself. A second 'grade' will be determined based on a more thorough check of your Python code submission. The grade for the whole Python assignment is a combination of both 'grades', computed by evaluating for which programming tasks you *both* demonstrated *and* submitted compliant code for.

1.2 Requirements on Source Code

The submitted *script source code* needs to fulfill the following requirements:

- Start from the template files include in the Debian system installed from Preseed, but also available separately in case you work on a different computer:
  `https://his.instructure.com/courses/8549/modules/items/199804`

  Check that the files are not corrupted by verifying that they have the following MD5 checksums:

Filename	Size in Byte	MD5 checksum
`access.log`	583528345	`46cc76e5118c036a37e4250f6a8d5edc`
`access-small.log`	217697	`108623624b3f2f7a45b0493c7abd41a2`
`existingaccounts.csv`	2187	`96000d0a0d4ee1301631d29e0daaa5dd`
`exjobb.csv`	46	`58495f7d799a9652bf9cee16b6b28cde`
`newstudents.csv`	82186	`7167a410a3279aea8a37e199b0c8453b`
`scriptprogramming-examination.csv`	51	`1a6539b1d78e12491288b5574363ba9c`
`single-examination.csv`	15	`e79bd3dfa348dd37599e550388e76076`
`users.csv`	63	`76e5f169de474d81d52880606c32a4f2`

  To compute a checksum, use the following command: `md5sum` *filename*

  Update `users.csv` by adding your own login and full name. Do not forget to submit this modified file along with your source code.

- All scripts must contain the following metadata as a comment close to the beginning. It has already been included in the template files provided to you.

```
#---------------------- METADATA ----------------------
# NAME: your full name
# USERNAME: your login name
# COURSE: Script Programming IT384G - Spring 2025
# ASSIGNMENT: Assignment 1 - Python - Task number
# DATE OF LAST CHANGE: date in ISO 8601
#------------------------------------------------------
```

  Do not forget to update the name, login, task number, and date before your submission.

- Your source code must contain relevant comments that explain your code. For non-trivial functions, use the mechanism to have documentation strings at the start of each function (multi-line comments with triple quotation marks). Using the same mechanism, document modules as well. Correct spelling and grammar is required. English is preferred, but Swedish is allowed.
- Your variables and functions must have (self-)descriptive names.
- Your scripts may not clutter up the output with error or debug messages. Scripts must run without Python-internal error or warning messages. Error conditions and exceptions, such as caused by invalid human input, must be handled gracefully unless otherwise specified.
- Scripts can be executed without requiring the teacher to make any modifications. For example, use relative paths for filenames when opening a file.
- All scripts must be able to run on the lab environment as used in this course.
- Your scripts are fully functional and in compliance to the provided instructions and requirements.
- Do not make use of variables that are shared between two Python files, for example a module and its test script. Communication between modules and other Python scripts shall exclusively happen via function invocation and its arguments and return values. Inside one Python file, you are allowed to declare and use variables as you see fit, both inside and outside of functions.
- Do not make use of modules not installed by default without prior approval by the main lab supervisor.
- Do not touch the filesystem, for example to open, read, or write to files, unless explicitly required by the instructions.

Good luck with the lab!

2 DOCUMENTATION ON PYTHON

For this assignment there is plenty of great documentation available for free. Here are some pointers:

- The course literature/material found on Canvas
- The official Python 3 documentation: `https://docs.python.org/3/`
- Real books used in this course, available for free to download from within the University network or via the library's proxy:
  - Beginning Python – From Novice to Professional
  - Python Recipes Handbook – A Problem-Solution Approach

Throughout the assignment, you may have to read or write CSV (comma-separated values, `.csv`) files. Python provides a module called `csv` to handle those files. Although it is not forbidden to use this module, it may be easier for you to use normal text file operations to read the file's content line by line and using the `split` function to split each line into columns. To write `.csv` files, you can use the normal `print` function and specify using the `file` argument to write into an open, writable file instead to writing on screen.

Read the complete instruction and consider how to design your scripts before starting to write code. A well-planned design will save you a lot of headache later.

3 TASK 1: COURSE GRADES

This course's course plan describes how the grades from the three separate examinations (Python, PowerShell, and written exam) get combined into a final grade for this course. The course plan explains how ECTS grades of each of the three examinations get translated into numbers, how the numbers get combined using a weighted arithmetic average, and how the result of the computation gets translated into the final grade.

Retrieve the course plan, which is available in English and Swedish, and read how the merger of grades is described. If you have problems understanding the formula, please talk to a lab supervisor.

English	`https://pdfproxy.his.se/coursesyllabus/121318/EN`
Swedish	`https://pdfproxy.his.se/coursesyllabus/121318/SV`

3.1 Merging Grades (Mandatory)

Your script `merge-grades.py` has to perform the following steps:

1. For each of the three examinations ('Assignment 1', 'Assignment 2', 'Written exam'), ask the user for an ECTS grade, i. e. either 'A', 'B', 'C', 'D', 'E', or 'F'. You can assume that the user will only enter one of the six valid values, so no error handling has to be implemented.

2. Internally, translate the letters into numbers as described in the course plan. There are several ways to realize that, so feel free to discuss alternatives with your peers or the supervisors. Eventually, you have to program your own, individual solution without re-using someone else's code.

3. Compute the final grade following the formula from the course plan. Research how to round the result of your computation to the closest integer number. For example, 2.4999 is to be rounded to 2, but 2.5 is to be rounded to 3.

4. Translate the resulting integer number back to a letter grade and present it to the user as the final course grade. If any of the three grades entered by the user is grade 'F', the resulting grade must be 'F' as well.

There is no requirement to organize the script's code into functions, but it is strongly recommended to do so to improve readability and to allow for re-use of code. Discuss the organization of code into functions with your peers or a lab supervisor if you need help with that.

Three separate example runs of the script with different input:

```
Assignment 1 grade: A
Assignment 2 grade: B
Written exam grade: C
Weighted final grade: B

Assignment 1 grade: E
Assignment 2 grade: E
Written exam grade: C
Weighted final grade: D

Assignment 1 grade: C
Assignment 2 grade: B
Written exam grade: F
Weighted final grade: F
```

Keyboard input is shown in italics in this document to visualize what is entered by the user, whereas the script's output is shown in upright characters. Your script shall not make use of italics or any other text formatting. If your Python code contains the string `\033`, you are doing something wrong.

3.2 Extended Requirements (1 Extra Point, Optional)

Enhance the script by adding the following features:

1. Add error handling when the user enters nothing (just presses enter) or anything that is not a letter grade. In such a case, print an error message notifying the user that the provided input does not meet the expectations. Then, ask again for a grade. If the user repetitively enters invalid input, print the error message and ask again for a grade each time, but exit the program with a non-zero exit code if the user enters invalid input three times in a row.

   ```
   Assignment 1 grade: K
   The value you entered is not an ECTS grade. Please try again.
   Assignment 1 grade: B
   Assignment 2 grade: B
   Written exam grade: C
   Weighted final grade: B

   Assignment 1 grade: G
   The value you entered is not an ECTS grade. Please try again.
   Assignment 1 grade: 3
   The value you entered is not an ECTS grade. Please try again.
   Assignment 1 grade: M
   You entered invalid values three times. Giving up.
   ```

2. Instead of assuming that there are exactly three examination weighting 2.5 hp each, your script shall read the examination structure from a `.csv` file. In this file, there are as many rows as there are examinations in the course (at least one). The file has two columns, separated by a semicolon (‘`;`’): the first column is the examination's name (for example ‘Assignment 1’), the second column is the examination's weight (for example ‘2.5’). Note that the course's total weight no longer needs to be 7.5 hp. Assume that all examinations continue to use the ECTS grading scale, so translating

between letter grades and numbers stays the same. The filename of the `.csv` file will be given as the single argument passed to this script.

For the script programming course, the file `scriptprogramming-examination.csv` looks like this:

```
Assignment 1;2.5
Assignment 2;2.5
Written exam;2.5
```

Test and verify that your script works with other potential `.csv` files like the following two examples (`single-examination.csv` and `exjobb.csv`, respectively):

```
Written exam;3
```

```
Project proposal;1.5
Report;27
Opposition;1.5
```

4 TASK 2: COMPUTING ACCOUNT NAMES FOR NEW USERS

Assume you work for the University's IT department and have to create accounts for new students. From the student administration, you get a list of student names and you have to create a unique account name for every student.

4.1 Account Name Construction (Mandatory)

The data you receive from the administration is a plain text file in UTF-8 encoding with one name per line. The names in each line consist of the last name, followed by a comma, followed by any non-negative number of whitespaces, followed by the first name. As the student administration is not really good with computers, the list may contain empty lines or whitespace surrounding names which you have to ignore. Below is an example for a list of new students as received from the student administration (a full list is available in `newstudents.csv`).

```
Jonasson, Anders
```

```
Lundgren, Elin
Lundqvist,Elin
Jönsson, Ann-Marie
Svensson,    Sven Olof
Gómez Iglesias, José Antonio
```

Assume that the names in the list use some Latin script (for example 'Akira', not 'アキラ'), but may contain non-ASCII letters like 'Ä', 'Å', or 'Ü'.

Account names your script is expected to generate based on students' names consist only of ASCII lowercase letters `a` to `z` and digits `0` to `9` and are always eight characters long. Non-ASCII letters must be transcribed to their ASCII base form, for example 'ä' to 'a'. You can either write your own solution which at least has to cover the letters 'ä', 'å', 'ö', 'ü', 'ó', and 'é', or use an existing Python module that implements this functionality. Punctuation marks like hyphens are to be removed. Do not forget to convert all letters into their lowercase form.

Account names are constructed as follows:

1. To avoid name clashes with identical parts for the remaining part of the account name, each account name starts with a 'conflict resolution string', which is a letter (`a` to `z`). Assume that you will not need more than 26 letters.
2. The year when the account was created. Do not hard-code the year, but rather use a Python date function to get the last two digits of the current year, for example `25`.
3. The first three letters of the first (given) name. Assume that the first name is always at least three letters long.
4. The first two letters of the last (family) name. Assume that the last name is always at least two letters long.

As the administration may supply you with lists of new students multiple times during a year, you have to check the existing student database to avoid generating student logins that already exist: there may be several students starting in the same year that have exactly the same name (or at least the same initial letters for their first and last names). Thus, when generating new login names, you have to make use of the conflict resolution letter to resolve any potential conflicts in names. In our simplified scenario, the student database is a text file with a list of existing student account names, provided as a separate file (`existingaccounts.csv`). An example may look like this:

```
a23davso
a25nikpe
b22arema
a24alewi
a25svesv
a24calfr
c23gusjo
a25leoek
```

Your script shall print out for each new student (based on the provided list of new students) the student's human name and the generated login name. The output may look like this:

```
Anders Jonasson              -> a25andjo
Elin Lundgren                -> a25elilu
Elin Lundqvist               -> b25elilu
Ann-Marie Jönsson            -> a25annjo
Sven Olof Svensson           -> b25svesv
José Antonio Gómez Iglesias  -> a25josgo
```

Your script must be structured by using functions to encapsulate distinct functionality.

There is no need to split the code into modules. Neither should your script write to or modify any file nor should it ask the user for any interactive input.

4.2 Extended Requirements (1 Extra Point, Optional)

Extend your script you programmed as follows:

- Extend your script to be able to correctly handle cases where the first name is shorter than three letters or the last name is just one letter. For example 'Bo' is a valid Swedish first name and 'O' is a Korean last name. Find a solution on your own, implement it, and clearly explain and document it in your code (in comments, for example). Your solution shall not violate the requirements on account names as stated above, including the length of eight characters.
- The five letters at the end of a login name may make up a word in English or Swedish which is considered offensive. For example, student Idida Ottman should not get `a25idiot` as login. Given the following list of five-letter sequences that are considered offensive, adapt your script to detect such words in potential login names and instead propose alternative login names that still follow the requirements for login names, are similar to the students' names, but are no longer offensive.

  ```
  idiot
  annbj
  jimha
  susbe
  ingfr
  marpe
  rolhe
  runli
  marsa
  cecho
  ```

  Note: the list above on purpose does not contain real five-letter offensive words to not teach you those words.

  You are allowed to hard-code this list in your Python code; there is no need to store it in a separate text file.

5 TASK 3: A SMALL GAME

Write a small game that allows the user to guess a number.

5.1 Game Description (Mandatory)

At the beginning at the game, ask the user for his/her login. From a user database, lookup the corresponding name for the entered login; see the implementation requirements below how to do that Then, greet the user by his/her real name if the login was found in the database.

At the beginning of the actual game, your script generates a random number between 0 and 1000 (but does not show it yet) to compare the user's input against during the game play. Then, repeatedly ask the user to guess a number in the same range; the number will be compared to the generated random number: If the user enters a number that is too large or too small, provide corresponding feedback, for example 'The number you entered is too small.' If the user enters the correct number or either 'q' or 'Q' (but not 'qqqq' or 'QUIT'), break from the loop but continue with the game otherwise

After breaking from the loop, either congratulate the user for winning or confirm that the game has been aborted. In either case, inform the user about all unique numbers sorted from smallest to largest he/she

has entered during this game. There is no need to ask the user if he/she wants to play again, simply let the script end.

Example game play where the user correctly guesses the number:

```
Hello, what is your login: maho
Hello Martina, welcome to this guessing game!
I will determine a random number between 0 and 1000 and you have to guess it.
You can quit at any time by entering 'Q'.
Please enter a number between 0 and 1000 or 'Q': 485
I am sorry, but this number is too small.
Please enter a number between 0 and 1000 or 'Q': 891
I am sorry, but this number is too large.
Please enter a number between 0 and 1000 or 'Q': 812
I am sorry, but this number is too small.
Please enter a number between 0 and 1000 or 'Q': 849
I am sorry, but this number is too large.
Please enter a number between 0 and 1000 or 'Q': 839
I am sorry, but this number is too small.
Please enter a number between 0 and 1000 or 'Q': 845
I am sorry, but this number is too large.
Please enter a number between 0 and 1000 or 'Q': 843
I am sorry, but this number is too small.
Please enter a number between 0 and 1000 or 'Q': 844
That is the correct number. Congratulations!
The unique numbers you have entered were: 485, 812, 839, 843, 844, 845, 849, ↵
  ↳ 891
Have a nice day!
```

Example game play where the user quits prematurely:

```
Hello, what is your login: maho
Hello Martina, welcome to this guessing game!
I will determine a random number between 0 and 1000 and you have to guess it.
You can quit at any time by entering 'Q'.
Please enter a number between 0 and 1000 or 'Q': 411
I am sorry, but this number is too small.
Please enter a number between 0 and 1000 or 'Q': 666
I am sorry, but this number is too small.
Please enter a number between 0 and 1000 or 'Q': 737
I am sorry, but this number is too large.
Please enter a number between 0 and 1000 or 'Q': 711
I am sorry, but this number is too large.
Please enter a number between 0 and 1000 or 'Q': 737
I am sorry, but this number is too large.
Please enter a number between 0 and 1000 or 'Q': Q
You have quit from the game.
The unique numbers you have entered were: 411, 666, 711, 737
Have a nice day!
```

Example user database (`users.csv`):

```
dmi;Doug McIlroy
maho;Martina
```

```
guwe;Gustav
erbe;Ervin
```

Your implementation, in addition to allowing the user to play the game as explained above, must fulfill the following requirements:

- To map user logins to user's full names, use the file `users.csv`. Do not forget to update this file by adding your own login and full name.

  The `users.csv` file is a plain text file with one line per user; the two columns (login and user's name) are separated by a semicolon ('`;`'). Your script may open this file only once for reading, so store its content in a dictionary for later use.

  If the login the user entered could not be found in the dictionary, your script shall greet the user with his/her login instead of his/her name:

  ```
  Hello, what is your login: sdfsbhwbuhrbh
  Hello sdfsbhwbuhrbh, welcome to this guessing game!
  ```

- Your script needs to handle any type of user input. If a user's input neither can be interpreted to a number between 0 and 1000 (inclusive) nor equals to 'q' or 'Q', print a short error message and ask again for input.

  ```
  Please enter a number between 0 and 1000 or 'Q': klbgafleighrlghirqwogl
  Your input does not look like a number between 0 and 1000 or 'Q'
  Please enter a number between 0 and 1000 or 'Q': 99999
  Your input does not look like a number between 0 and 1000 or 'Q'
  Please enter a number between 0 and 1000 or 'Q':
  ```

- Collect numbers entered by the user in a suitable data structure. Note that you should only track unique numbers. In the example above, the user entered 737 twice, but this number is listed only once at the game's end.

- Write code to nicely format the list of entered numbers. It is not allowed to just let Python 'dump' the data, for example with `print(listvariable)`

5.2 Extended Requirements (1 Extra Point, Optional)

To earn one extra point, enhance the game by the following feature:

If the user had guessed the right number (and not prematurely quit from the game), the user may enter a high score list. For that, maintain a high score list file that records the ten best game runs over all time, where 'best' is the lowest number of guesses until reaching the correct number. The high score list file is a plain text file where the format is as follows: one line per game with two columns separated by a semicolon: login (not human name; for example `maho` instead of `Martina Holmberg`) and number of guesses.

To use the high score list file (`highscore.csv`) after the game, proceed as follows:

1. If it exists, open the high score list file and read the content into an appropriate data structure.
2. If the high score list is not already 10 elements long or if the just-finished game's score is better than one or more games recorded in the high score list, insert the just-finished game at the right position into the data structure. Drop any records beyond the 10th entry.
3. Write back the updated high score list to the same file.

Thus, open the high score list file once for reading and once for writing.

Show the high score list to the user and notify him/her whether he/she has entered the high score list and at which position. The list should be printed nicely formatted into columns. Again, translate login names to human names where possible.

Example game play (only the last few lines are shown):

```
I am sorry, but this number is too large.
Please enter a number between 0 and 1000 or 'Q': 333
That is the correct number. Congratulations!
The unique numbers you have entered were: 124, 194, 333, 401, 462, 584, 643, 733
You required 8 guesses which made you enter the high~score list.
| Doug McIlroy   |   4 guesses |
| Martina        |   7 guesses |
| Martina        |   7 guesses |
| Martina        |   8 guesses | < YOU
| Gustav         |  11 guesses |
| Gustav         |  13 guesses |
| Martina        |  15 guesses |
| sdfsbhwbuhrbh  |  21 guesses |
| aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa| 23 guesses |
Have a nice day!
```

Example high score list file

```
dmi;4
maho;7
maho;7
maho;8
guwe;11
guwe;13
maho;15
sdfsbhwbuhrbh;21
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa;23
```

6 TASK 4: REGULAR EXPRESSIONS

Earlier statements about the module's functions and their arguments and return values applies here as well.

6.1 Evaluating Apache Logfiles (Mandatory)

Write a module called `apachelog` (file `apachelog.py`) that provides the following function.

`aggregatelog(filename)` takes a filename to an Apache web server log file (for example `access.log`) and returns a dictionary with key-value pairs. The keys refer to two fields in a log entry: IP address and HTTP method. The values are again dictionaries where the keys are the unique values found throughout the log file and the dictionaries' values are the count of occurrences.

A simplified example for the return value could be the following, where 16 lines of log data got processed, where 15 lines document GET requests and one line documents a POST request.

```
{
  'IPaddress': {'127.0.0.1': 5, '192.168.1.1': 11},
  'Method':    {'GET': 15, 'POST': 1}
}
```

Make use of regular expressions as supported by the `re` module to evaluate each line in the log file and to extract the requested data. Make use of at most two invocations of `re` methods like `search` or `match` per input line to extract IP address and HTTP method, respectively. Do not use the `split` function or any other text string search or extract function to retrieve either method or IP address from a text string.

We provide two log files:

- `access.log` is a large log file containing 2647213 lines of log data. Eventually, your script must be able to handle this file (during the demonstration), but during the development and testing of your script you want to use the smaller variant below. A good script implementation should take less than one minute to process this file.
- `access-small.log` is a random selection of 1000 lines from above file. This makes it much faster to test your script to see if it is working correctly. Some exotic cases, such as rare HTTP methods, are missing in this log file, though.

Your script shall open and read an Apache log file only once. Loop only once over the log file's lines to collect the data as detailed below.

Your test script (file `apachelog-test.py`) shall evaluate the returned dictionary and print out:

- All encountered HTTP methods like `GET` or `POST` with their number of occurrences, ordered from most popular to least popular HTTP method.
- The single most popular IP address and its number of occurrences
- The number of unique IP addresses in the log file, i. e. the number of IP addresses that *occur at least once in the log file*, not the number of the IP addresses that occur exactly once in the log file.

Neither compute this data in `apachelog.py` nor return it via function `aggregatelog`.

Expected output for `access-small.log`:

```
Method      GET :     681
Method     POST :     306
Method     HEAD :      13
Most popular IP address is 198.50.156.189 with 93 occurrences
Number of unique IP addresses: 358
```

Expected output for `access.log`:

```
Method      GET : 1985985
Method     POST :  619230
Method     HEAD :   41831
Method      PUT :      70
Method PROPFIND :      70
Method  OPTIONS :      27
Most popular IP address is 198.50.156.189 with 167812 occurrences
Number of unique IP addresses: 51103
```

6.2 Extended Requirements (1 Extra Point, Optional)

To earn one extra point, make changes to your Apache logfile analysis scripts while maintaining backwards compatibility, meaning function `aggregatelog` works and behaves as described above despite the changes you make here.

Implement the following changes to your Apache logfile analysis module (file `apachelog.py`):

- Whereas you were allowed to use *multiple* regular expressions to evaluate log message lines or to ignore data in log message lines that were neither HTTP methods nor IP addresses, now you have to write *one, single, large* regular expression that analyzes a log message line completely. Use named capture groups for the following fields from a log message line:

  `IPaddress` IP address as before

  `DayOfMonth` Day of month

  `Month` Three-letter abbreviation for month, e.g. `Jun`

  `Year` Four digit string for year

  `Hour` Two digits, starting with either `0`, `1`, or `2`

  `Minute` Two digits, starting with `0`, …, `5`

  `Seconds` Two digits, starting with `0`, …, `5`

  `TimeZone` Offset for time zone, e.g. `+0200`

  `Method` HTTP method as before

  `Path` Path requested via HTTP

  `HTTPversion` Requested HTTP version, e.g. `HTTP/1.1`

  `HTTPstatus` HTTP status code, e.g. `200`

  `RequestSize` Size of request, sequence of digits

  `Referrer` URL or a single dash (‘–’)

  `UserAgent` String identifying the web browser or a single dash (‘–’)

- Enhance the return value of `aggregatelog` to return a more complex dictionary that now contains key and values for the extracted values.

Enhance your test script (file `apachelog-test.py`) to also print out the single most popular value for each of the added fields and the number of occurrences.

Expected output for `access-small.log` below the output provided by the original script:

```
  IPaddress : 198.50.156.189                          with      93 occurrences
 DayOfMonth : 21                                      with     105 occurrences
      Month : Jun                                     with     249 occurrences
       Year : 2018                                    with     464 occurrences
       Hour : 7                                       with      75 occurrences
     Minute : 40                                      with      29 occurrences
     Second : 5                                       with      27 occurrences
   TimeZone : +0200                                   with     707 occurrences
     Method : GET                                     with     681 occurrences
       Path : /administrator/index.php                with     311 occurrences
HTTPversion : HTTP/1.1                                with     966 occurrences
 HTTPstatus : 200                                     with     618 occurrences
RequestSize : 4498                                    with     212 occurrences
   Referrer : -                                       with     354 occurrences
  UserAgent : Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like
    ↪ Gecko with     277 occurrences
```

Expected output for `access.log` below the output provided by the original script:

```
  IPaddress : 198.50.156.189                          with  167812 occurrences
 DayOfMonth : 27                                      with  377419 occurrences
      Month : Jun                                     with  853777 occurrences
       Year : 2018                                    with 1678351 occurrences
       Hour : 7                                       with  198978 occurrences
     Minute : 50                                      with   58462 occurrences
     Second : 1                                       with   47509 occurrences
   TimeZone : +0200                                   with 2028455 occurrences
     Method : GET                                     with 1985985 occurrences
       Path : /apache-log/access.log                  with 1153953 occurrences
HTTPversion : HTTP/1.1                                with 2570464 occurrences
 HTTPstatus : 200                                     with 1303279 occurrences
RequestSize : 4498                                    with  415058 occurrences
   Referrer : http://www.almhuette-raith.at/apache-log/ with 1059200
    ↪ occurrences
  UserAgent : Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like
    ↪ Gecko with 1028045 occurrences
```

If you are not getting the exact numbers as shown above, one reason may be that your matches-complete-line regular expression does not match some lines and thus those lines' methods, dates, or user agents do not get counted. To address this issue, for debugging purposes, let your script print out those lines from the log file where your regular expression does *not* find a match. Then, manually check and compare those printed lines against your regular expression to see why those lines were not matched. Eventually, your script should produce the numbers like above and no non-matched line should be printed.

A CHANGES TO THIS DOCUMENT

Here are some changes made to this document after it was released at the beginning of the course. Changes may be required to clarify ambiguities or correct errors.

Apr 29
- Replaced `IPaddr` with `IPaddress` for Apache Log task.