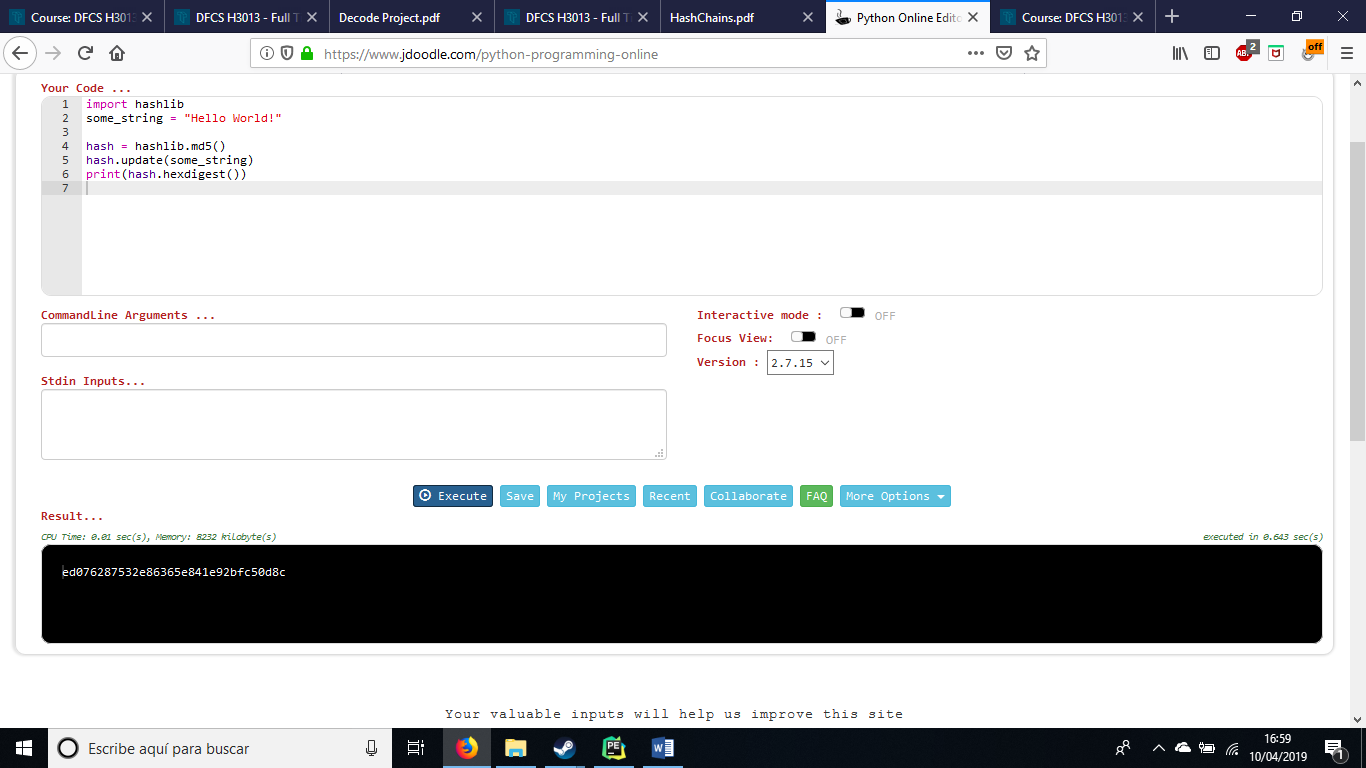
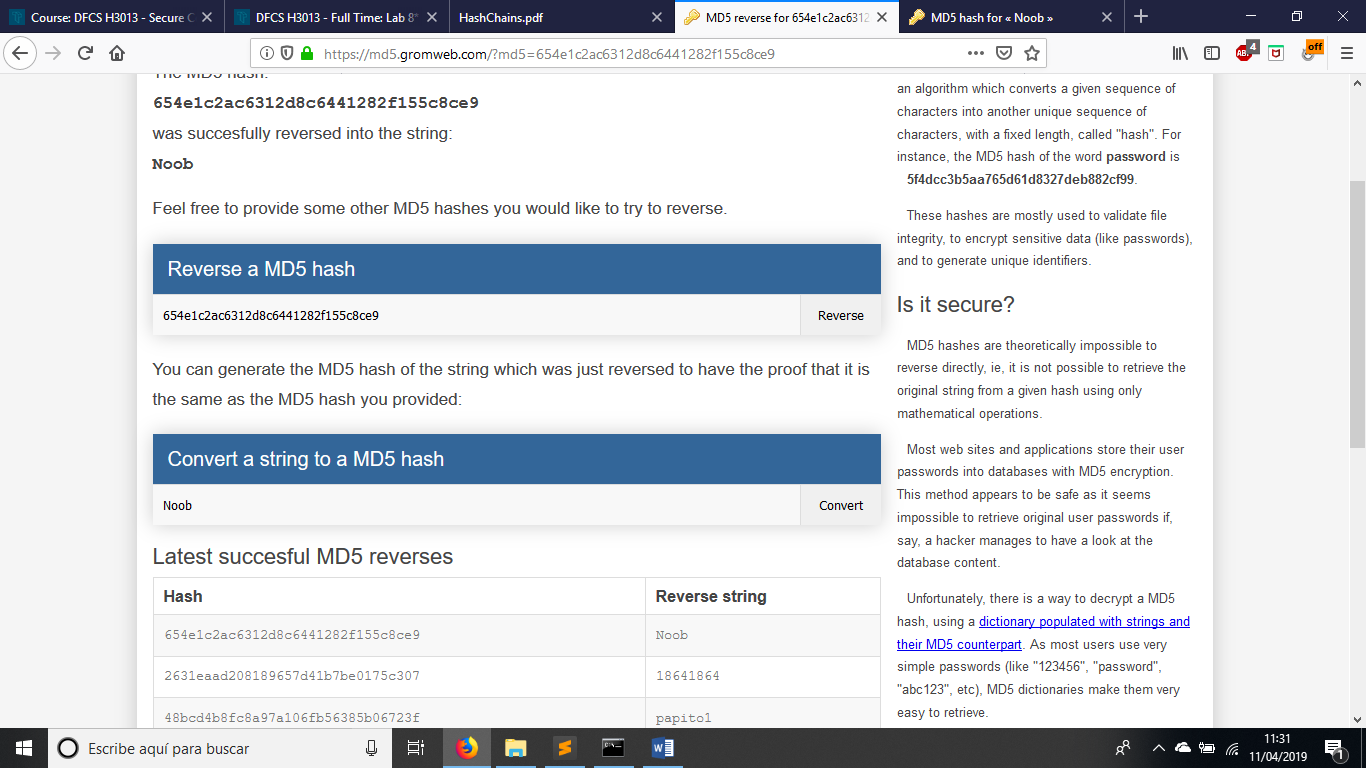
Lab 8 - BlockChain

Cristian Villarroya Sánchez

The first test is just for testing how to show a hash in python2.



Now its time to search for info, we have the root for login as the user nOOB. If we search for it on google, we have a match for that key. Is a md5 key and the match is the word Noob.



If we look the new word and the original word, we can see that the upper and lower cases are swapped. This gives as a hint about how to calculate the root.

Knowing this, I have create a method in python that, given a seed, just swap the lower case for upper case and viceversa.

Now its time to calculate the hash, but first lets do a little bit of research about hash chains.

A hash chain is simply a hash of a hash of a hash…

They can be used in login processes. For example, if we hash 1000 times, the server will store the hash 100 of the password which is provided by the user. Then, when the user authenticates, it sends the hash that hashes that hash in the server (hash99(password)).

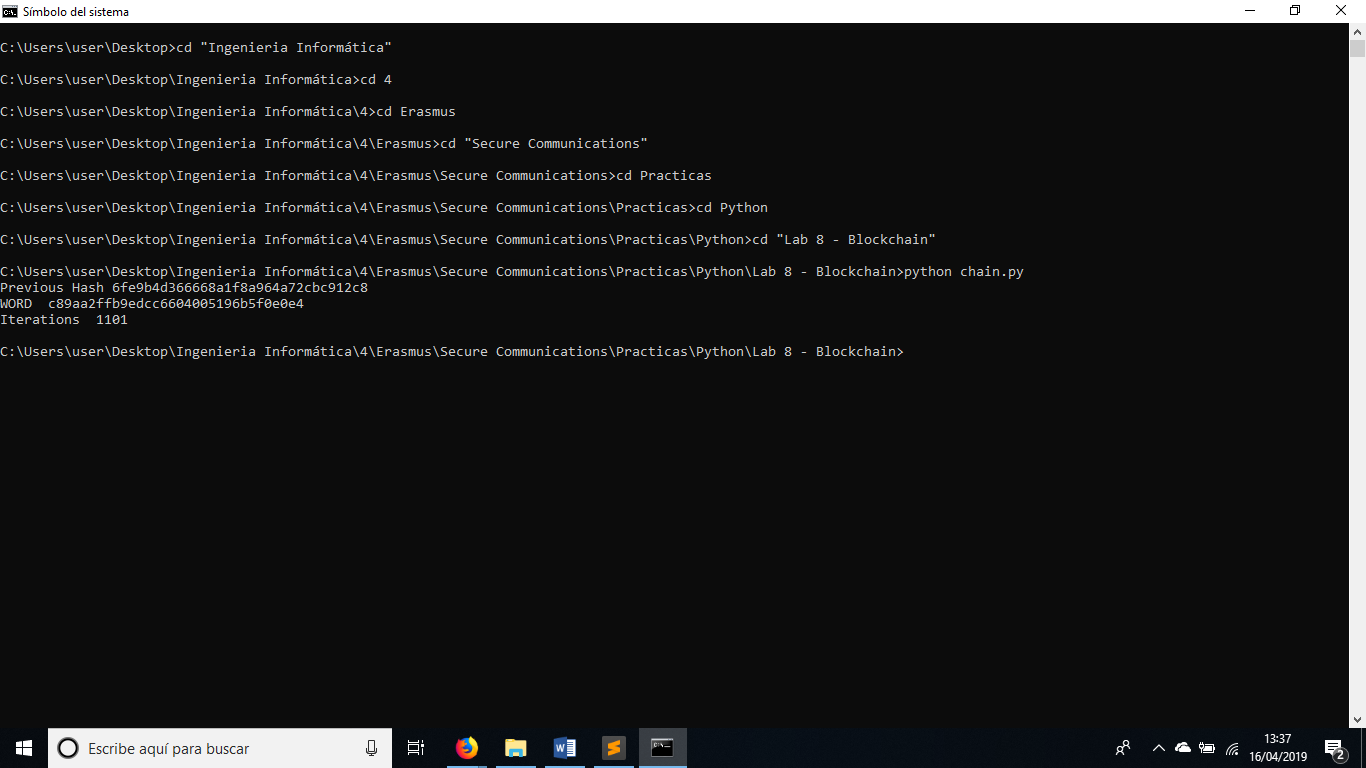
Knowing this, we are asked for the hashX-1 of the password that hashing that value will result in the given hash (c89aa2ffb9edcc6604005196b5f0e0e4)

For this, I have create a method in python. This method calculates a hash from a given seed.

Now that I have the methods for calculating the seed, and calculating the hash I can start looking for the hash asked in this exercise.

The method is simple, I star with the seed given (ECSC), then I calculate the seed (swap lower-upper cases) and I create a first hash of it. I have the expected hash stored in a value. I also create a tmp value for storing the previous hash(that will be the solution of the exercise) and a value to know how many iterations have been made. Finally, I create a while method in which I iterate until the word(hash) is equal to the expected hash. When the match, I print the previous hash(solution), the actual hash (expected) and the iterations made.

Here is the final output :



So the solution will be 6fe9b4d366668a1f8a964a72cbc912c8 which can be found in 1101 iterations.