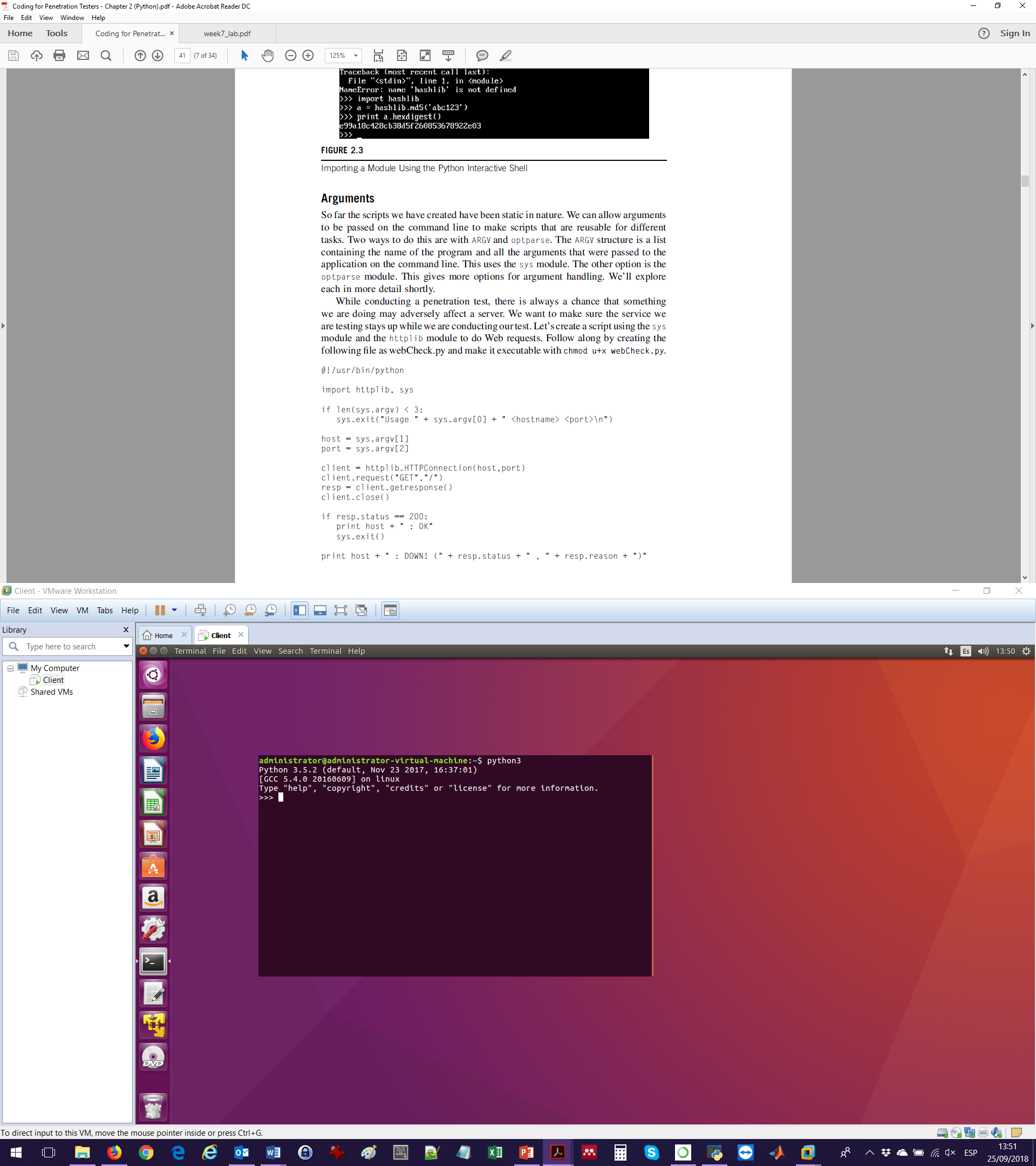
**Hashlib**

Open a terminal

Install Python 3 (sudo apt-get python3)

Install pip sudo apt install python-pip. This program will allow you to install python modules easily.

Verify that Python 3 has been installed correctly by writing python3. This will run Python



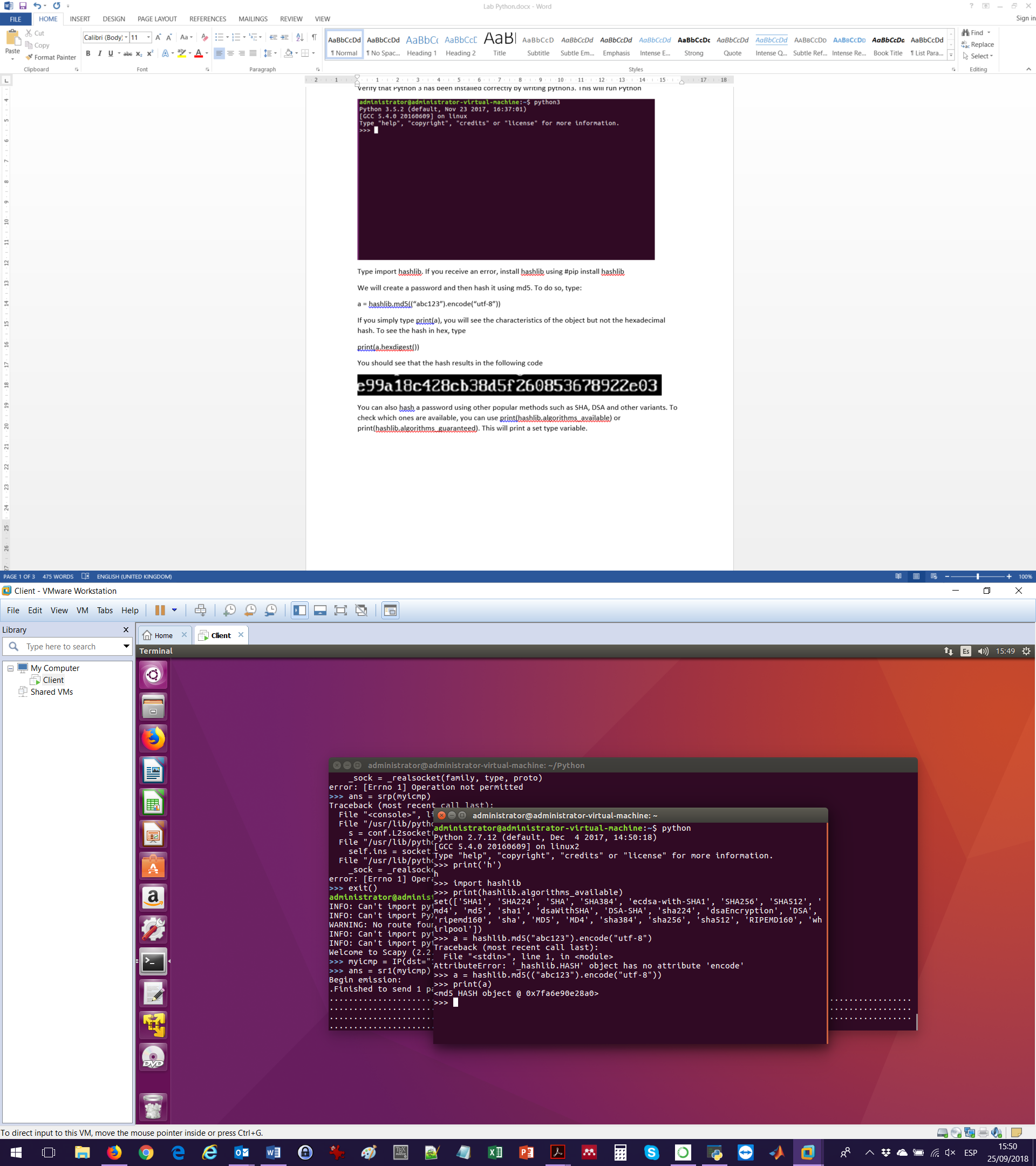
Type import hashlib. If you receive an error, install hashlib using #pip install hashlib

We will create a password and then hash it using md5. To do so, type:

a = hashlib.md5((“abc123”).encode(“utf-8”)) or a = hashlib.md5((“abc123”).encode()) or a = hashlib.md5(b“abc123”)

Notice that instead of specifying the encoding, by putting a “b” before the string the function will know that the string has to be converted into bytes before being hashed.

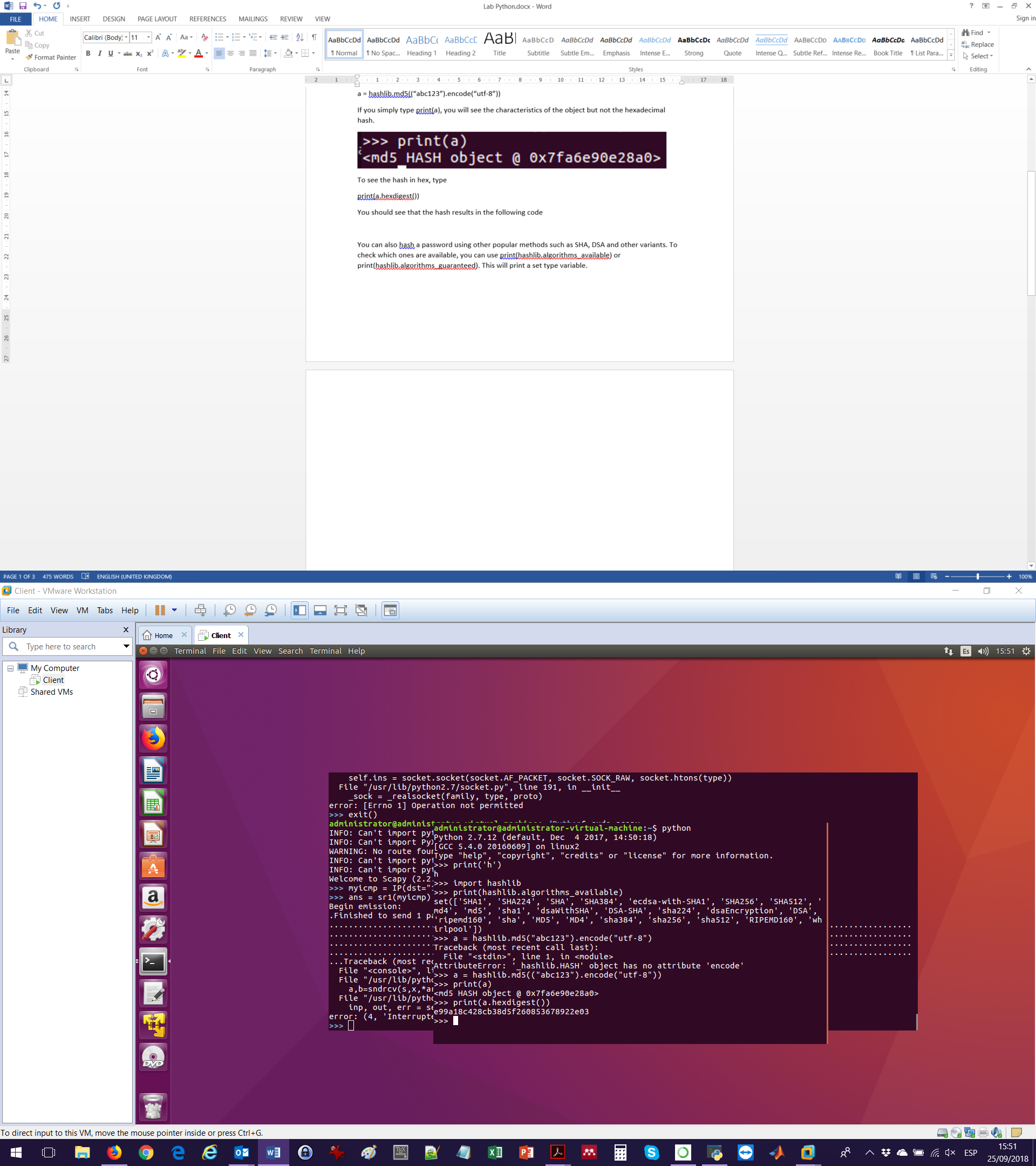
If you simply type print(a), you will see the characteristics of the object but not the hexadecimal hash.



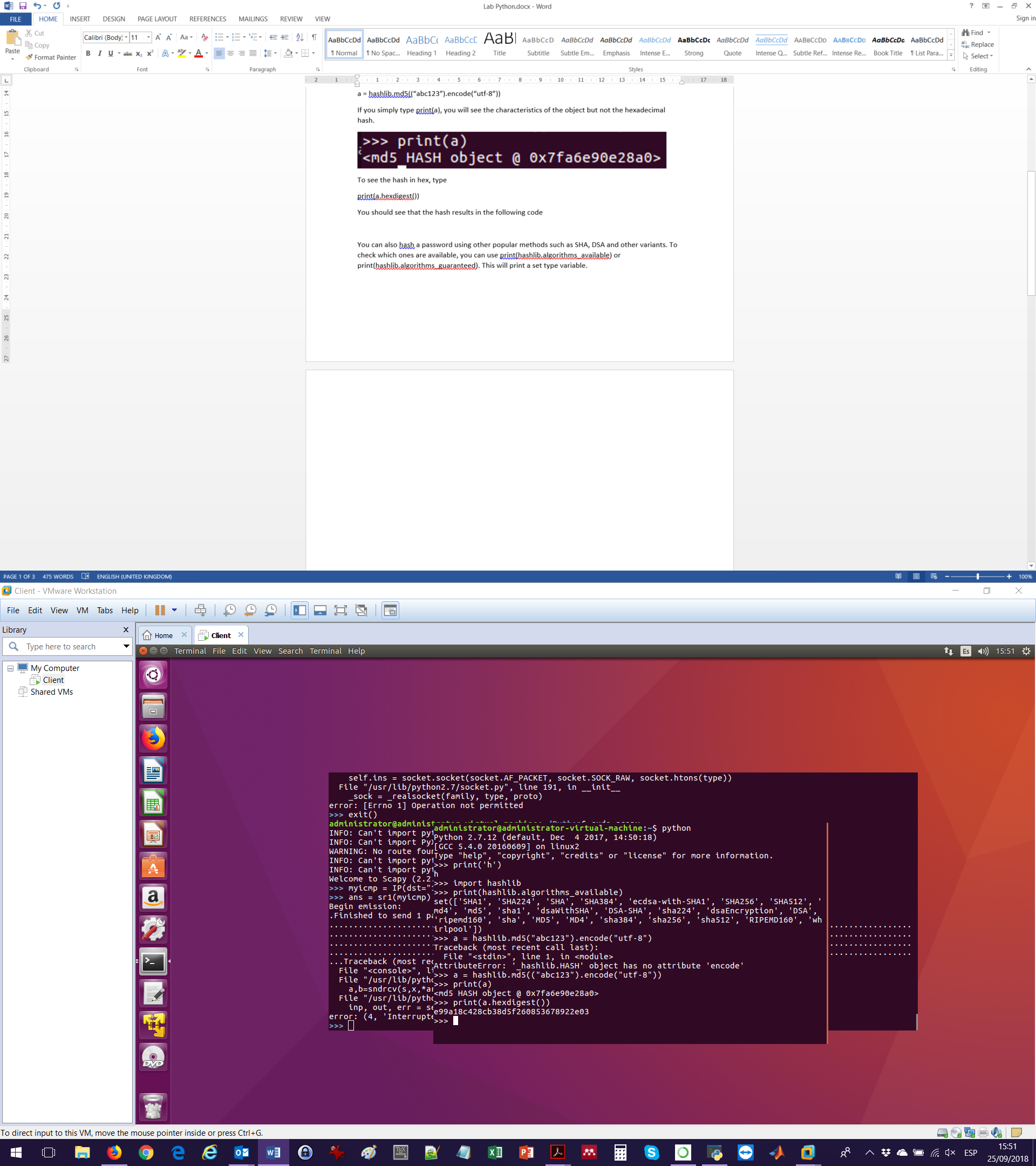
To see the hash in hex, type

print(a.hexdigest())

You should see that the hash results in the following code

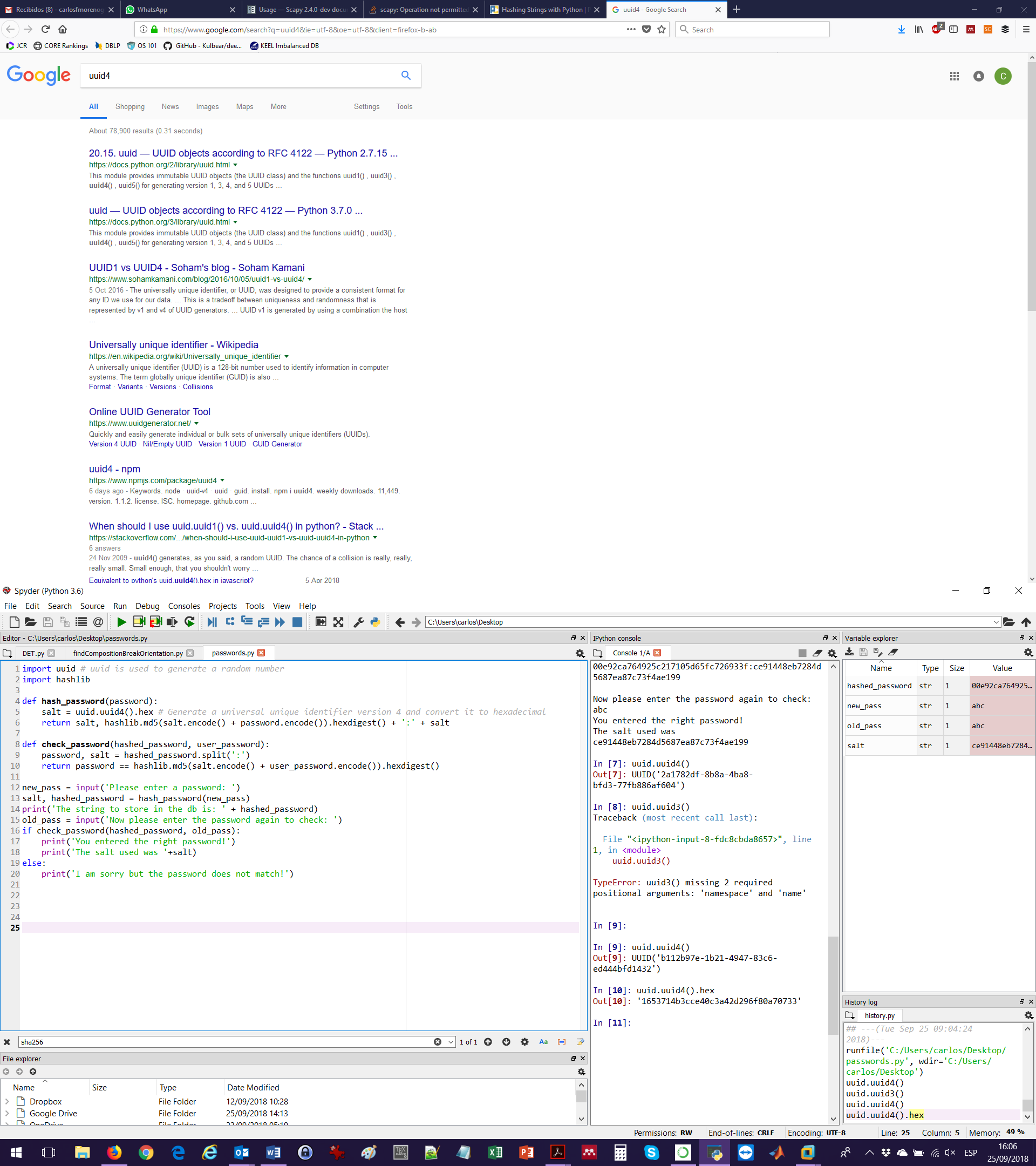


You can also hash a password using other popular methods such as SHA, DSA and other variants. To check which ones are available, you can use print(hashlib.algorithms\_available) or print(hashlib.algorithms\_guaranteed). This will print a set type variable.



Now that we know how to hash a password, you will write a script where you will create a password, and then verify the correctness of this password. To enhance the security of the password, we will use a salt, which is a ransom sequence added to a password before hashing. The salt is used to prevent dictionary attacks.

Open gedit and write the following function. Save it as “passwords.py”



The script passwords.py needs two python modules: hashlib and uuid (universal unique identifier). Then, it defines two functions:

1. hash\_password(): This function takes a string “password” as an input. Then, it creates a random salt in hexadecimal, and finally it returns a hexadecimal value composed of the password plus the salt, separated by the “:” character. Notice that instead of defining a variable to store the hash and returning the value of such variable, in Python it is possible to put the operation after the “return” (line 6), saving lines and time.
2. check\_password(): This function takes the stored password “hashed\_password” and the password tried by the user “user\_password”. The function first splits the hashed password into the actual password and the salt by using the “split” function that is aware that the “:” separates both elements. Finally, the function will return a True or False value: either the stored password is equal to the new one (the user password has to be hashed to do this comparison) or it is not. Notice that once again the operation of comparison is being used after the “return” (line 10).

After the two functions have been declared, the main program is executed (line 12). First, the password to store is requested to the user by using the “input” function. Then, “stored\_pass” is hashed and the salt is appended (line 13). Afterwards, the user is presented with the hash of the password to store (line 14). The user is immediately requested for the password; this new password is stored in the variable “new\_pass” (line 15). Then, the function “check\_password()” is called to verify if the password is correct or not (line 16). Notice that it is not necessary to write the whole condition “if check\_password(hashed\_password, new\_pass)==True”; when checking an if, the default value to accept the condition is TRUE. If the password is correct, the user gets shown an approval message and the salt (line 17). Otherwise, the user gets an error message.

BONUS: Modify the script to allow the user 3 attempts for the password. In addition, allow the user to select between “MD5”, “SHA1” and “DSA” hashing functions (TIP: You can either create different python functions for each hashing function or use an if/case structure within the python functions).

**Scapy**

The Scapy module provides advanced packet manipulation using the Python framework. It allows to manipulate and process packets at every layer of the Open Systems Interconnection (OSI) stack.

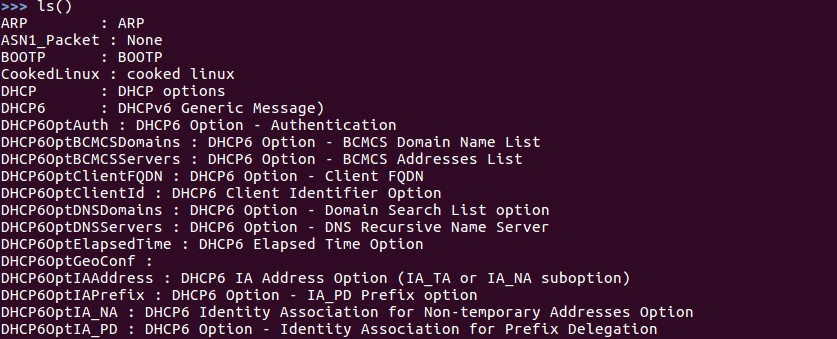
First we are going to look at the Scapy interactive shell to help make things a bit more familiar when we move into scripting with Scapy. We will build packets to figure out how to navigate the interactive shell, and then we will work on scripts to handle two penetration testing problems: transferring data over Internet Control Message Protocol (ICMP) and processing sniffed data.

Scapy’s homepages are [www.secdev.org/projects/scapy/](http://www.secdev.org/projects/scapy/) and <https://scapy.readthedocs.io/en/latest/index.html> . Here you can find documentation, training opportunities and download the latest version of Scapy.

To install scapy, write sudo apt install python-scapy

To run scapy type sudo scapy. Notice that it has to be run using root privileges. For the moment, don’t worry about the warnings and info received.

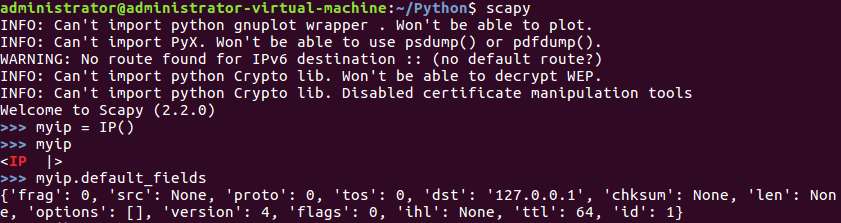
To see all the types of packets that can be created, press ls()



To create an IP packet, first we use the IP function IP() and assign the packet to a variable called myip by. To see the default values of this packet, type

myip.default\_fields

This will return a dictionary variable.



To send a basic ping, we need to fill the IP information and create an Internet Control Message Protocol (ICMP) packet. First, we check the ICMP fields by typing ICMP.default\_fields. Then, we create a packet which will stack IP and ICMP layers. To do so, we can use the “/” operator. Type the following command myicmp = IP(dst="192.168.1.1")/ICMP(type="echo-request") to store the ping request in the variable myicmp.

To send and receive packets, there are three options:

* sr(): Sending packets and receiving answers.
* sr1(): Variant that only returns one packet that answered the sent information. Only used for layer 3 packets (IP, ARP, etc…)
* srp(): Variant of sp1() for layer 2 packets (Ethernet, 802.3, etc…)

To send our packet, we will use sr1(). Type the following command:

ans = sp1(myicmp)