**Python**

**What is Python?**

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic link; make it very attractive of Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python´s simple, easy to learn syntax emphasizes. Python supports module and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

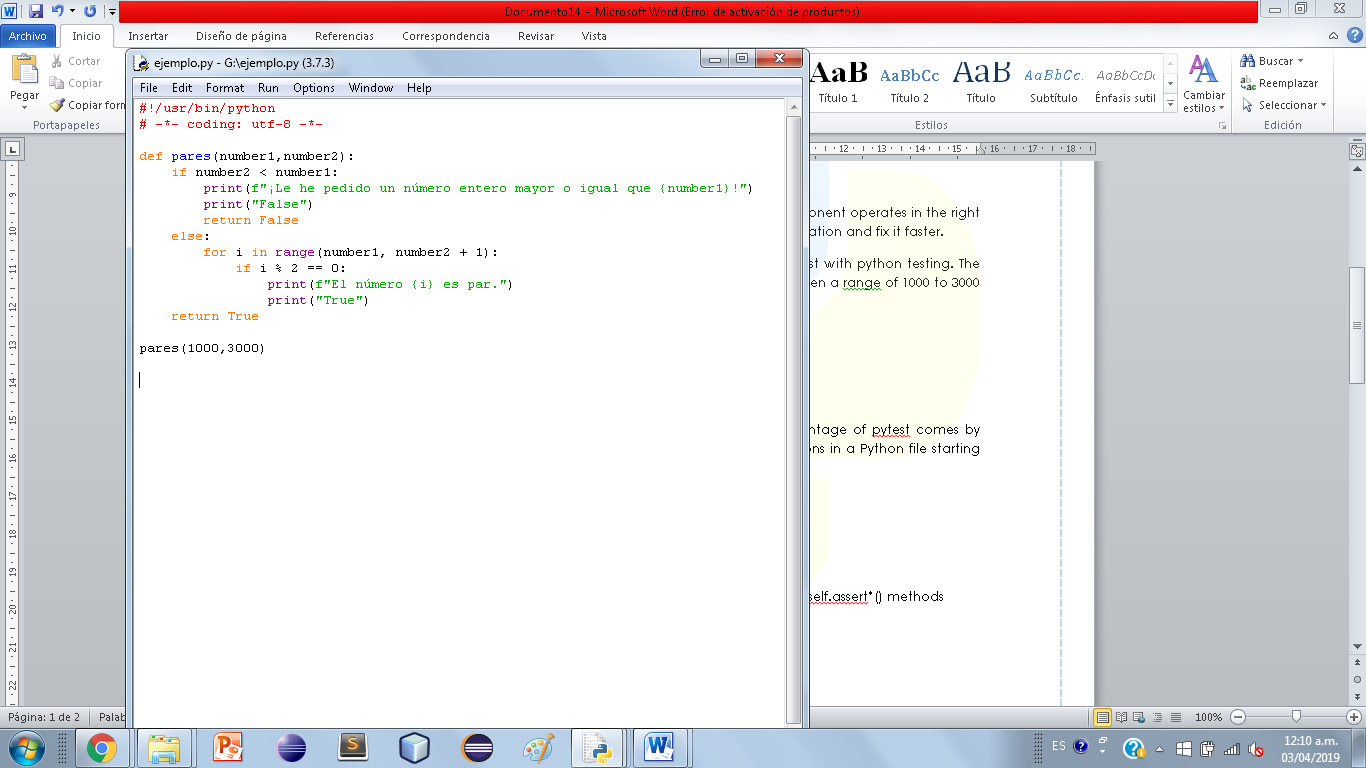
**Python Test**

**Getting Started With Testing in Python**

A unit test is a smaller test, one that checks that a single component operates in the right way. A unit test helps you to isolate what is broken in your application and fix it faster.

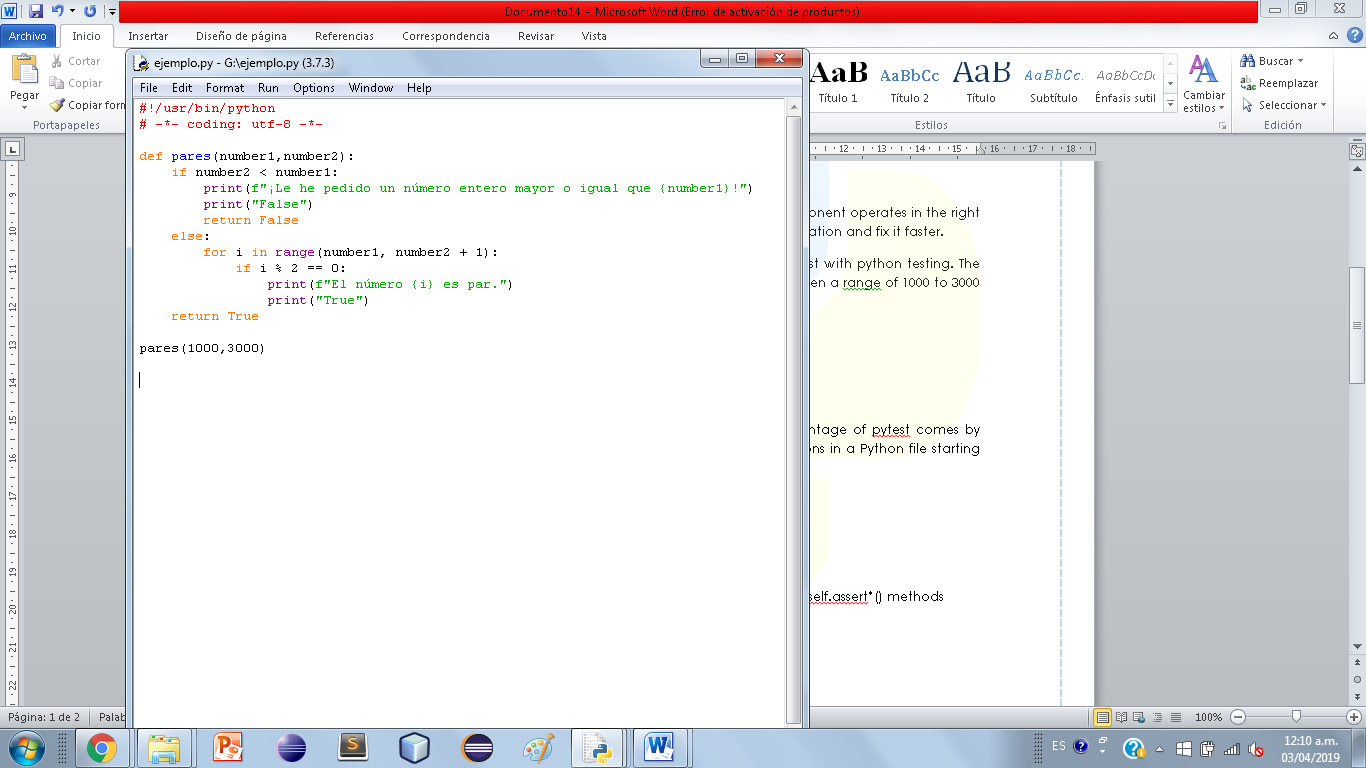
In this report we will begin by explaining how to perform a test with python testing. The operation of the program is very simple; all even numbers between a range of 1000 to 3000 will be shown.

The principal code is:

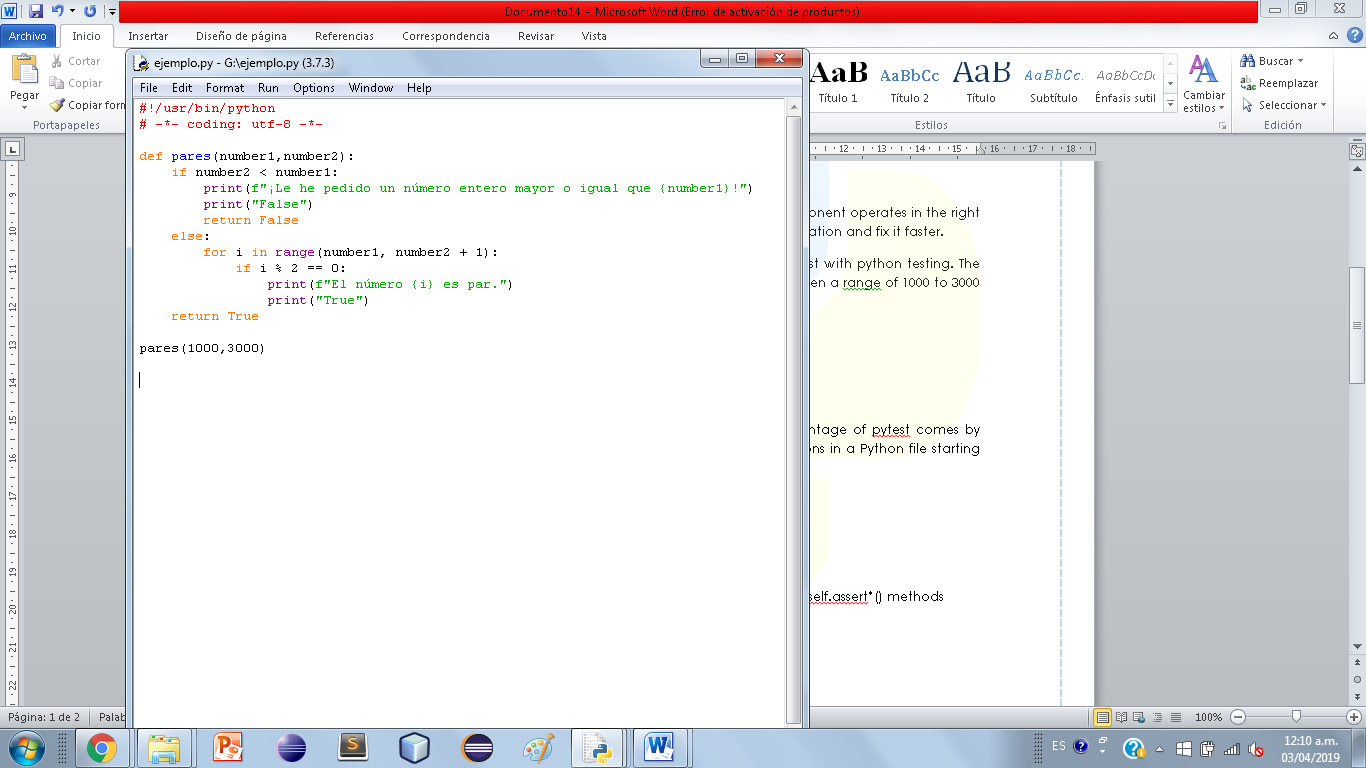


**Explaining the code:**

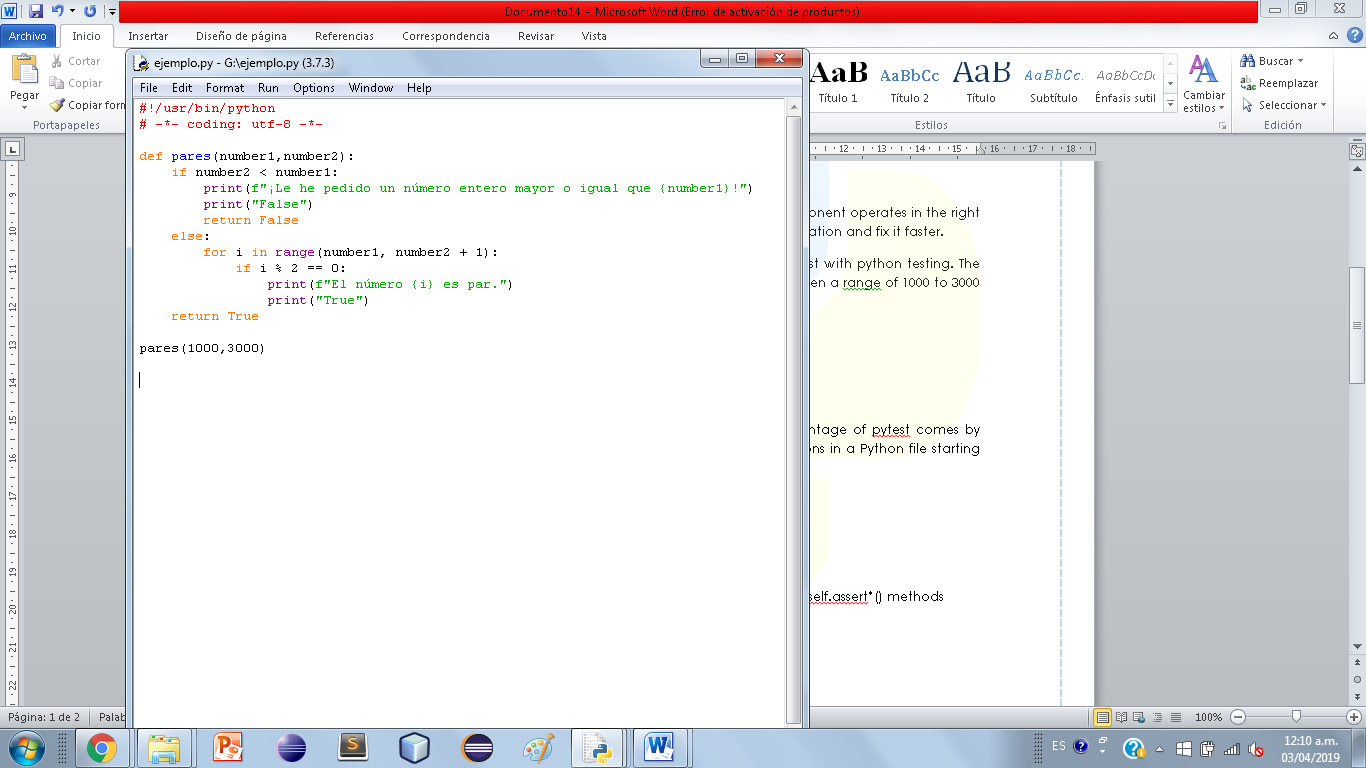
First we must import the python libraries.



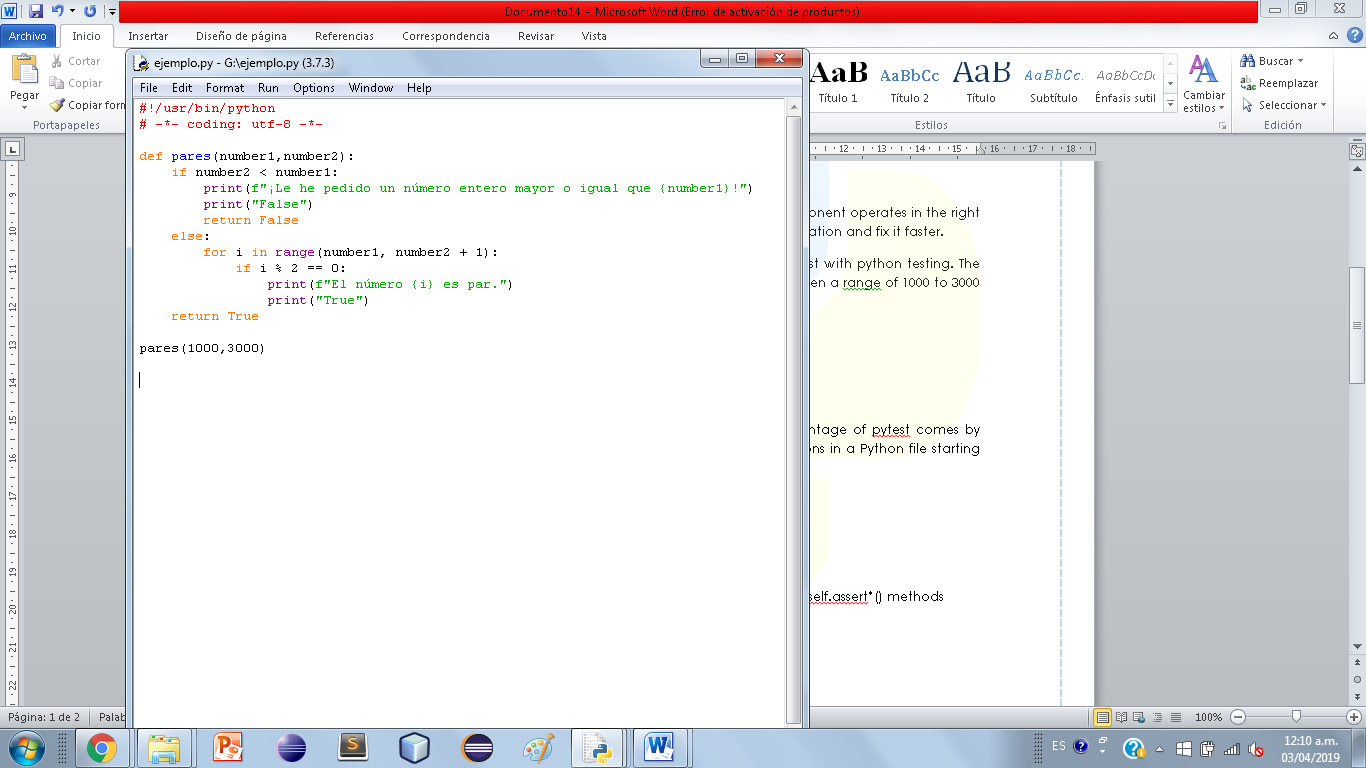
Then we define the variables called number1 and number2.



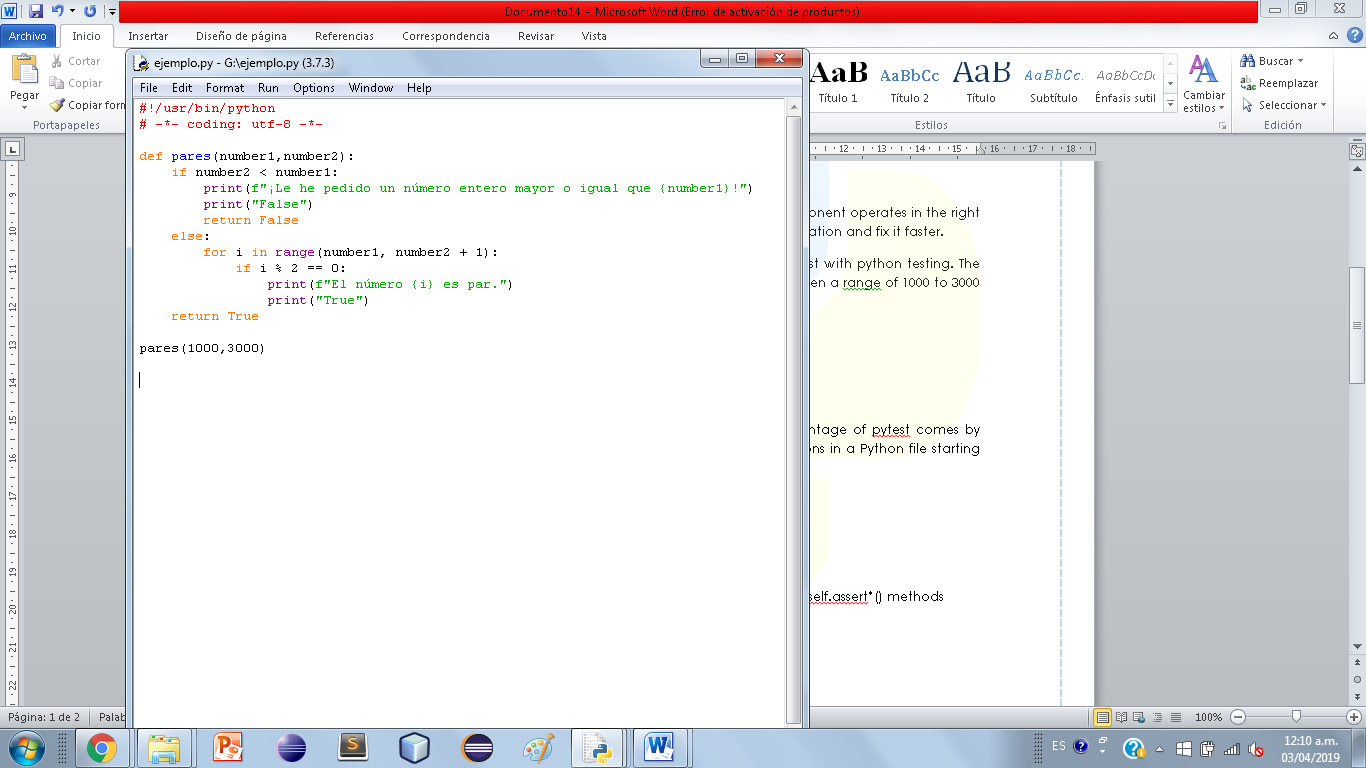
Since we define the variables, we perform the if statement, where we make the comparison, indicating if the number2 is less than number1, it will show the message on the screen indicating that we need to buy a greater value and not less in the value number2, and return false.



If it is not this way then it will enter the else, where we assign a for with a value I that will be all the even numbers generated in a range, we generate our process that says the number1 entered and the number2 + 1. If so, the values I it is going to be divided between 2 and it must be == 0, this is the remainder in a division, if it shows zero it means that it is an even number. If everything is correct, it will print all the even numbers between that range, and it will return true.



To finish the main class of the code, we declare the vales of the rank that we want to generate their even numbers, mentioning that it will be between 1000 and 3000.



**Pytest**

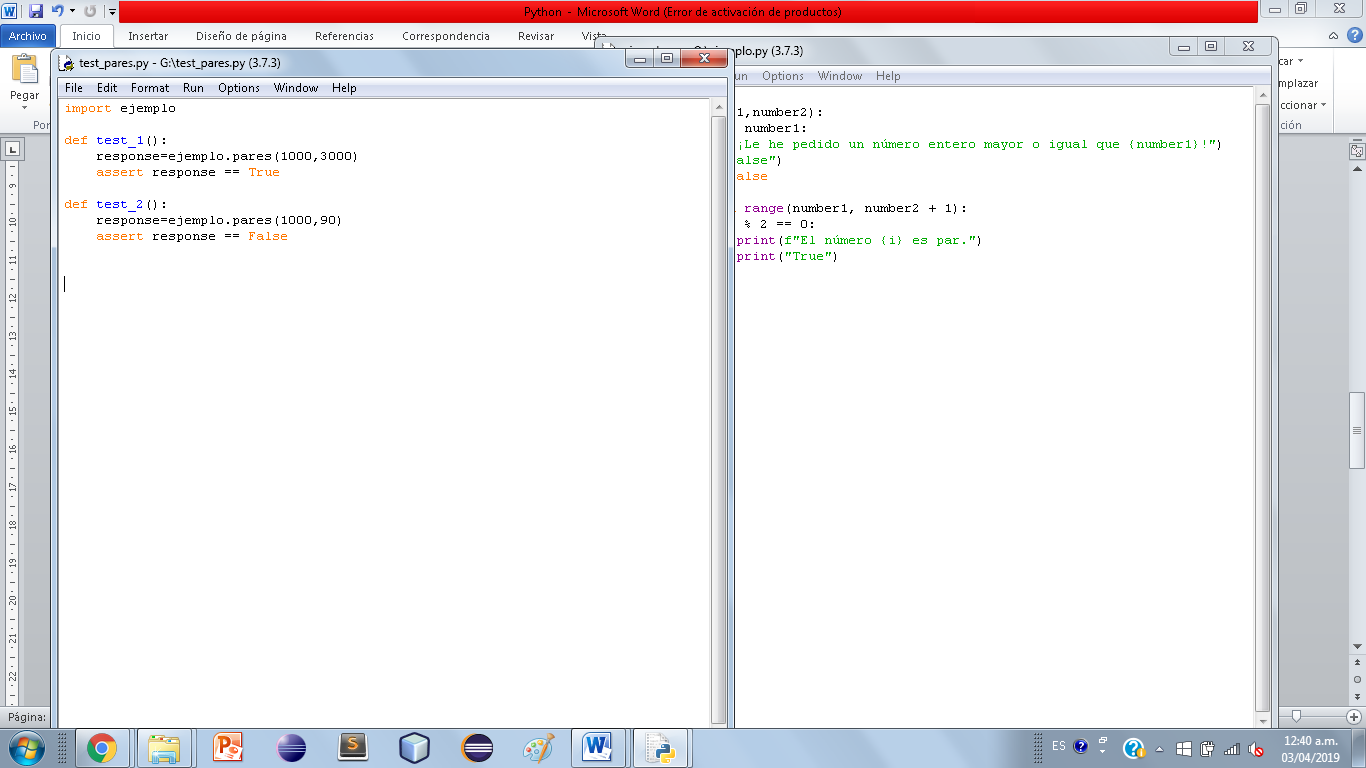
Pytest supports execution of unittest test cases. The real advantage of pytest comes by writing pytest test cases. Pytest test cases are a series of functions in a Python file starting with the name test\_.

**Pytest has some other great features:**

\*Support for filtering for test cases

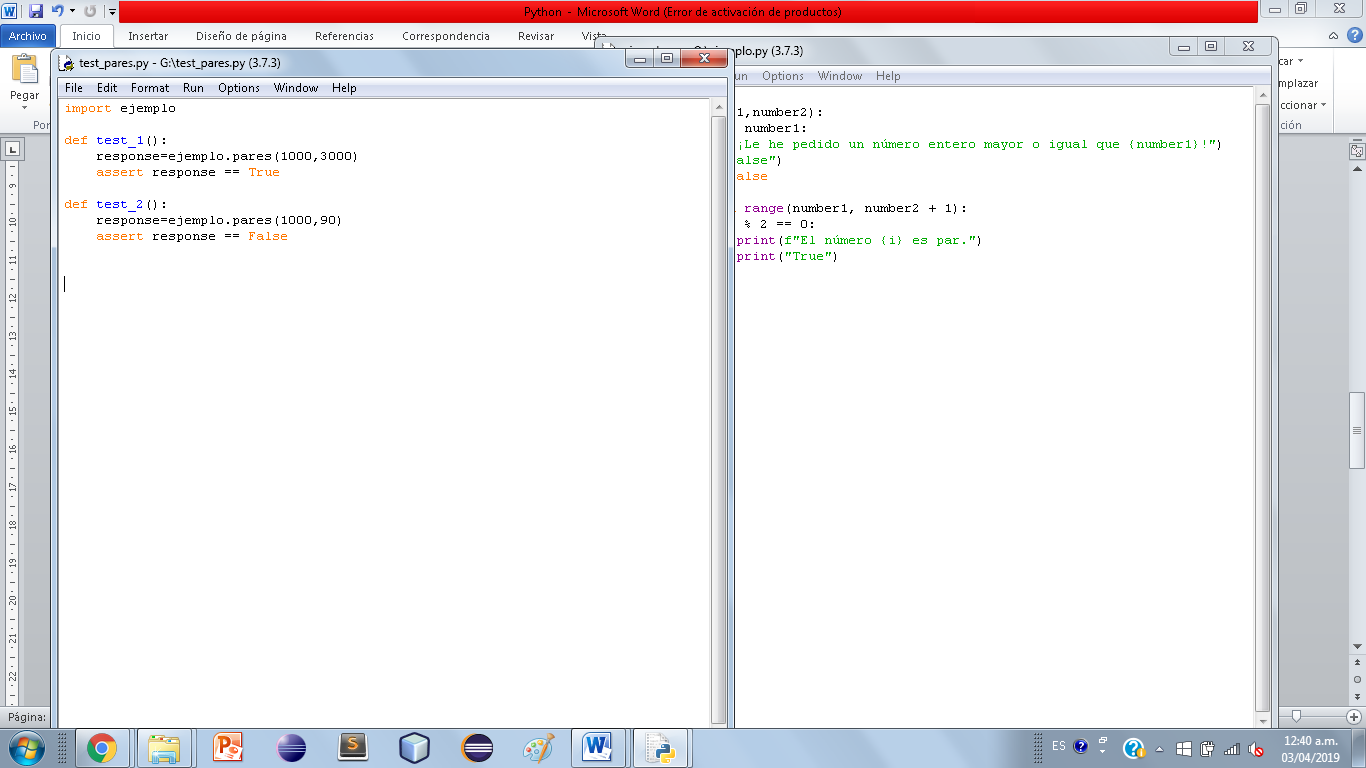
\*Ability to rerun from the last failing test

Writing the test\_pares test case example for pytest would look like this:



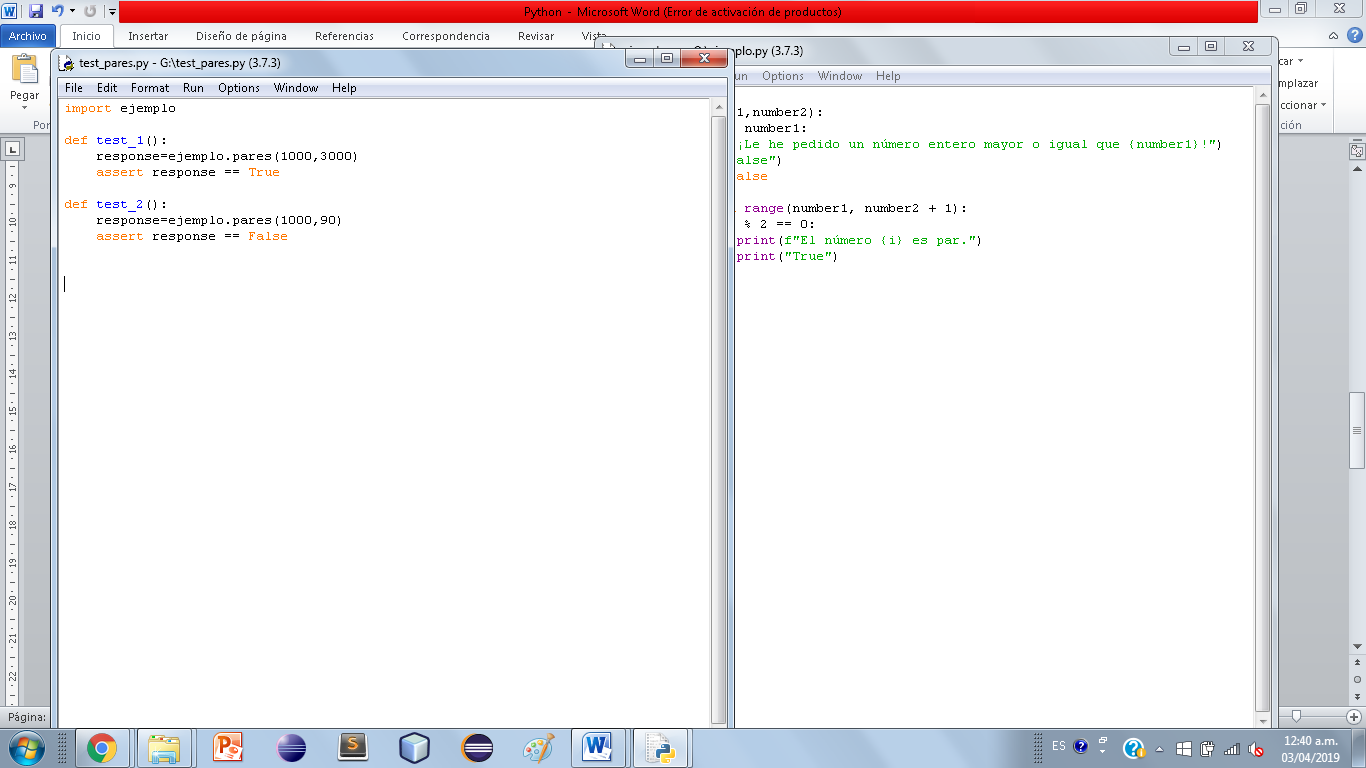
**Explaining the code:**

First we must import the main class of the code.

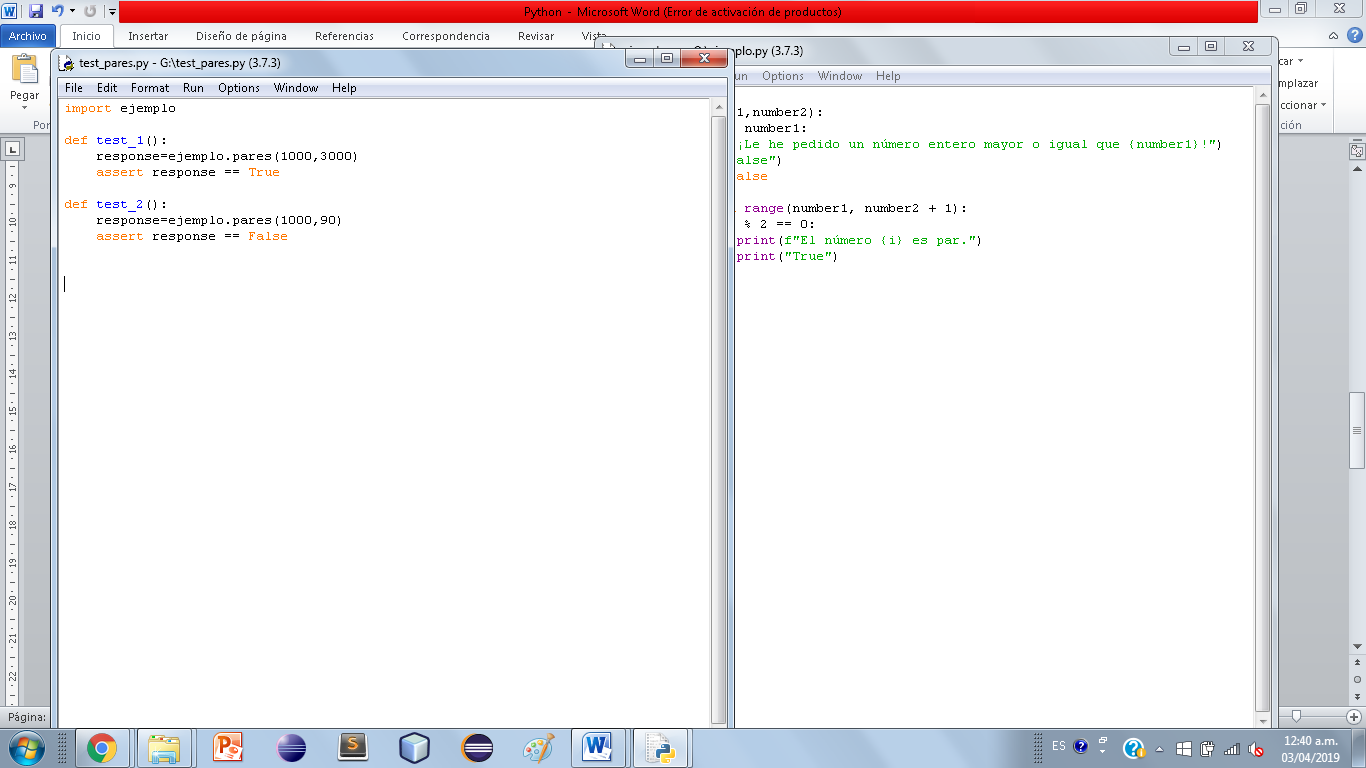


After we declare the methods for each testing that will be generated, tests are performed for each function that will be executed.

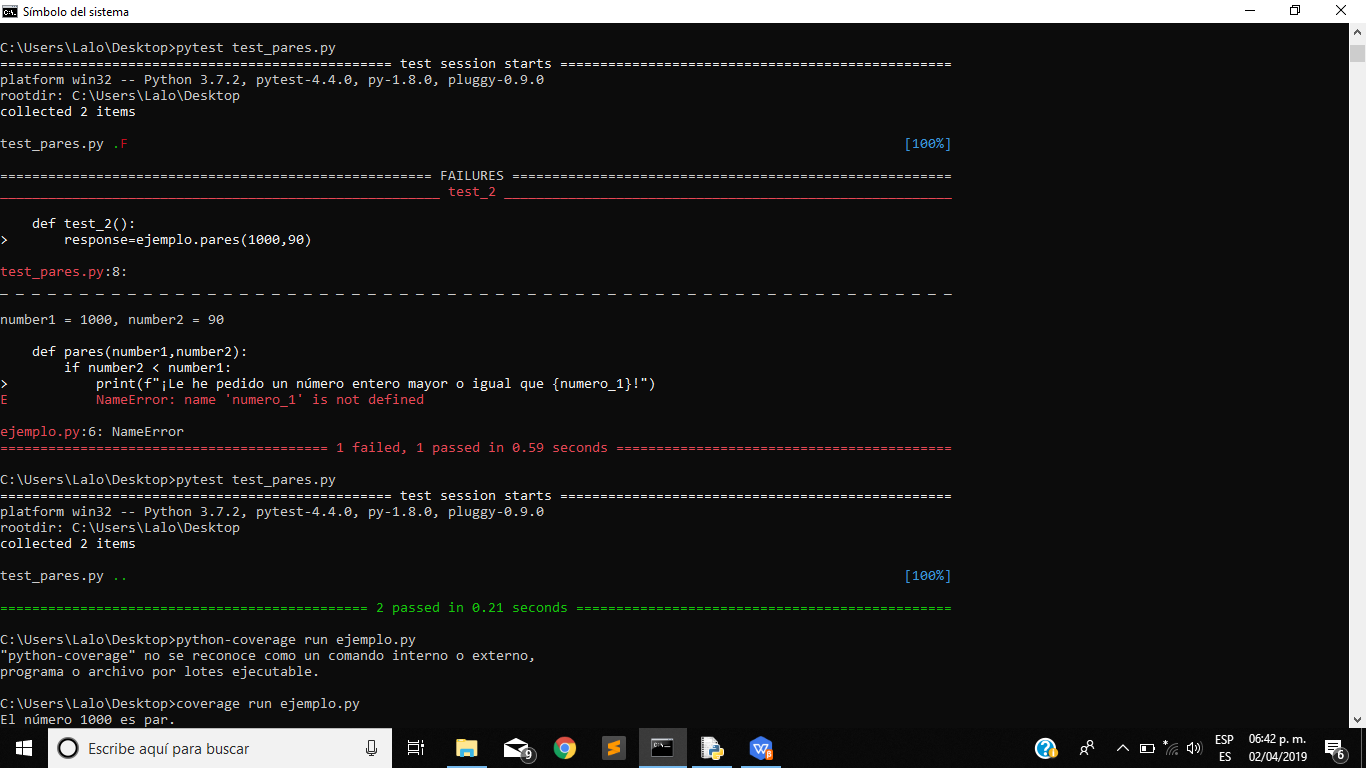
First we define the first test\_1 (): that in the main class is when the function is true. In this method it indicates that if the main code (example) we enter from 1000 to 3000. I will be true, because we are indicating the number2 greater than number1.



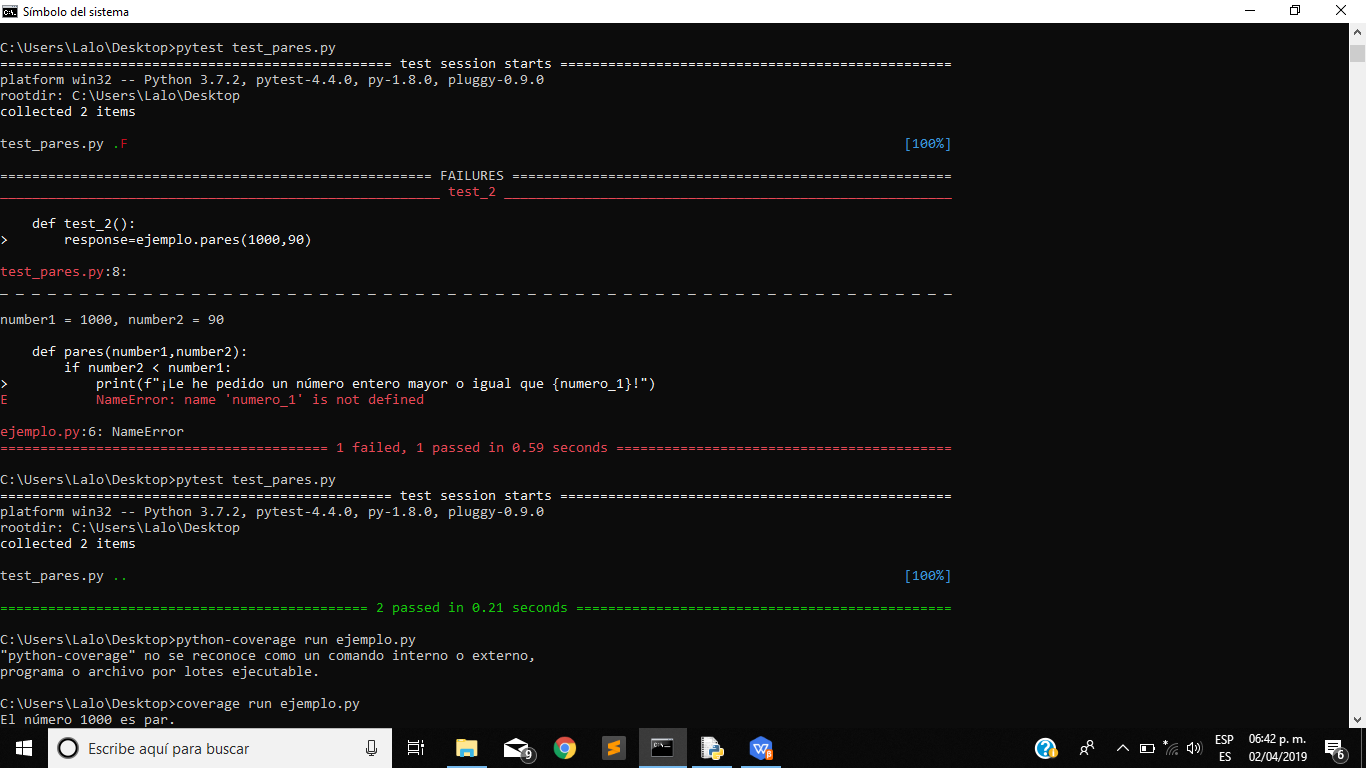
Then we define the second test\_2 (): that in the main class is when the function is false or erroneous to what is asked mainly. In this method it indicates that if the main code (example) we enter a smaller number in the value of number2 as it shows from 1000 to 900. I will be false, because we are indicating the number2 must be greater than the number1.



When we enter the pytest to perform the code testing. If the code has an error, it will show the error in the testing. Here the error is that we write the variable incorrectly when ordering it in the print.



Now if we run the program without any errors, we will see that in the testing it already generates the code without any problem.



**Coverage**

Coverage.py is a tool for measuring code coverage of Python programs. It monitors your program, noting which parts of the code have been executed, then analyzes the source to identify code that could have been executed but was not.

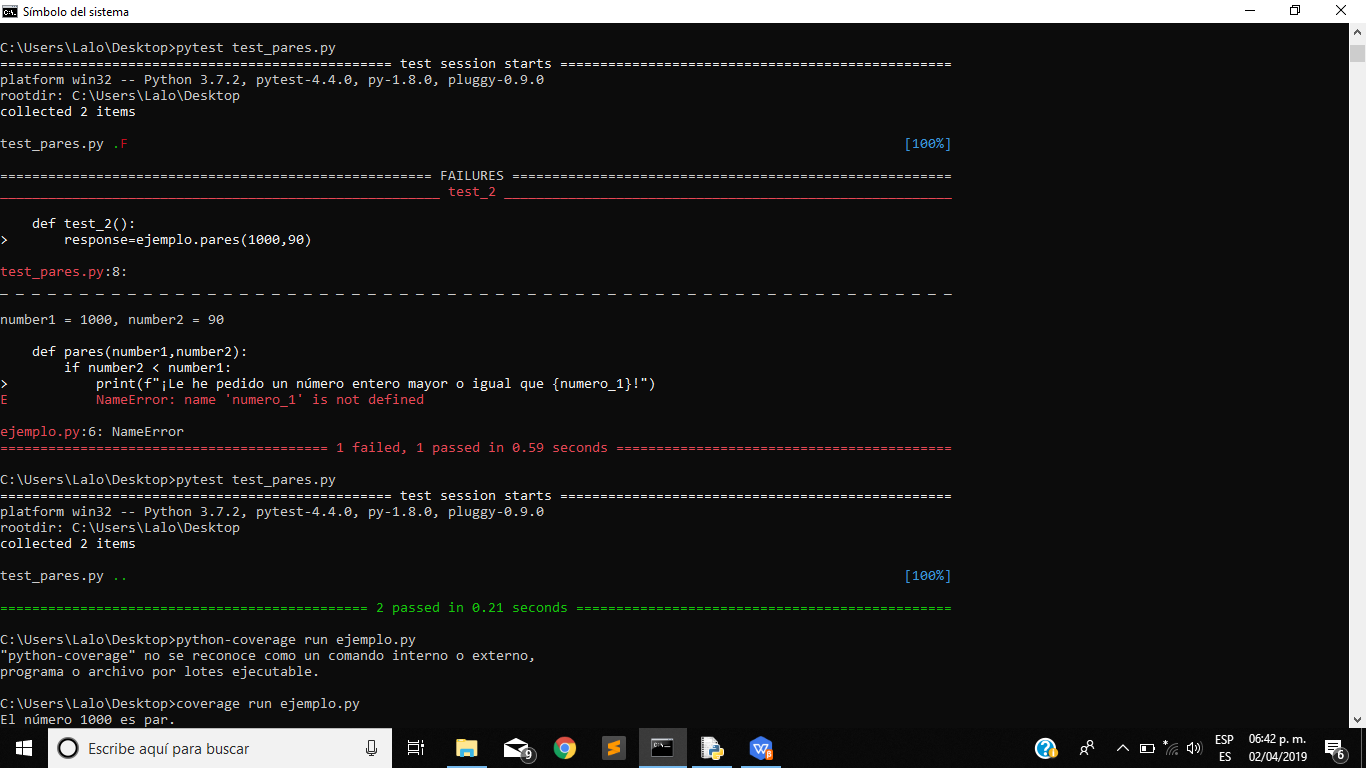
Coverage measurement is typically used to gauge the effectiveness of tests. It can show which parts of your code are being exercised by tests, and which are not.

There are a few different ways to use coverage.py. The simplest is the command line, which lets you run your program and see the results. If you need more control over how your project is measured, you can use the API.

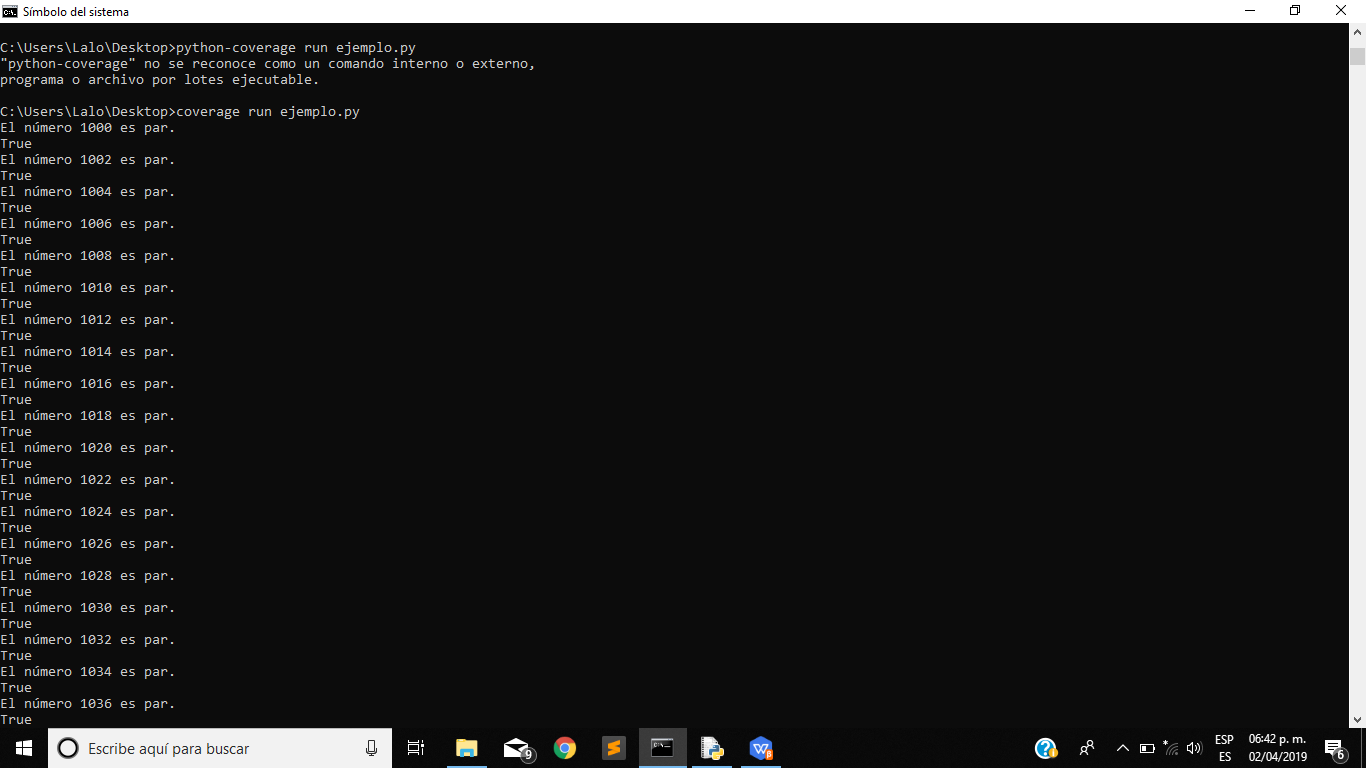
Some test runners provide coverage integration to make it easy to use coverage.py while running tests. For example, pytest has the pytest-cov plugin.

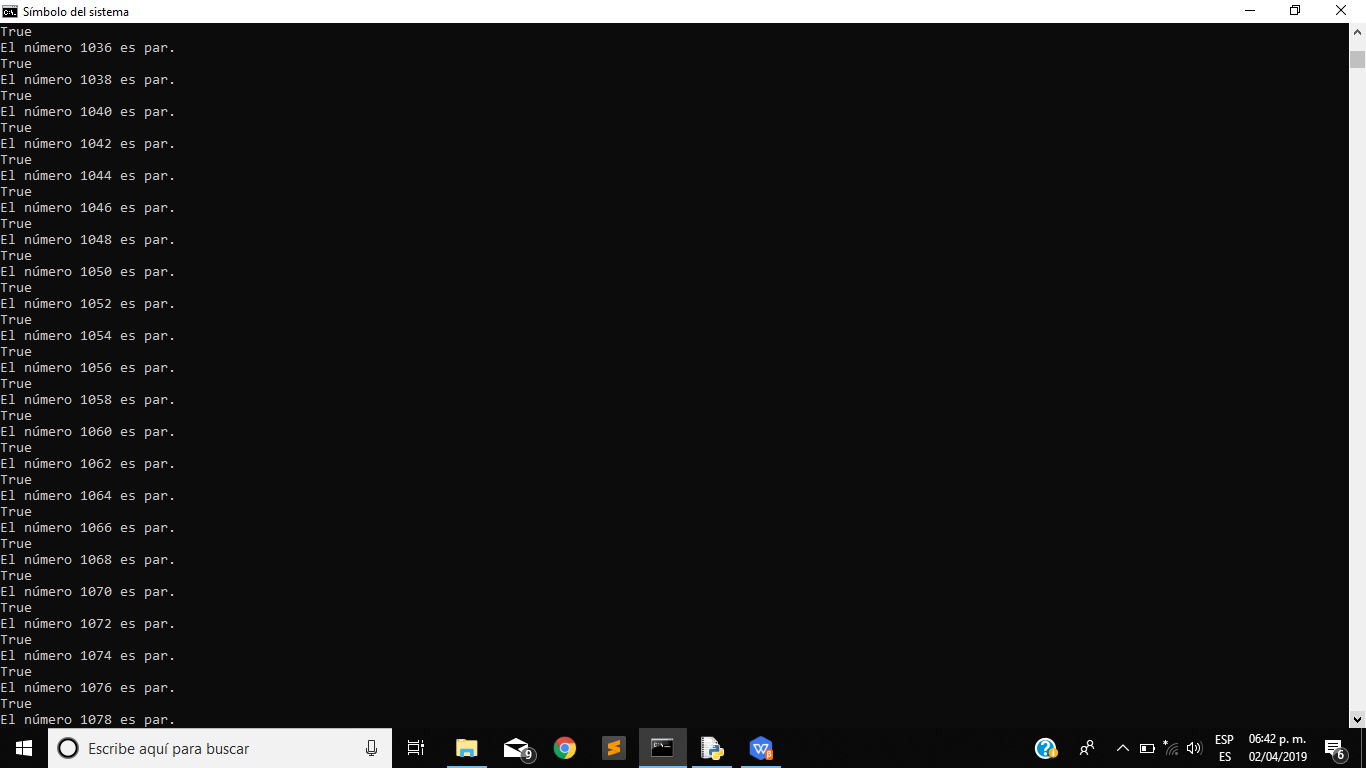
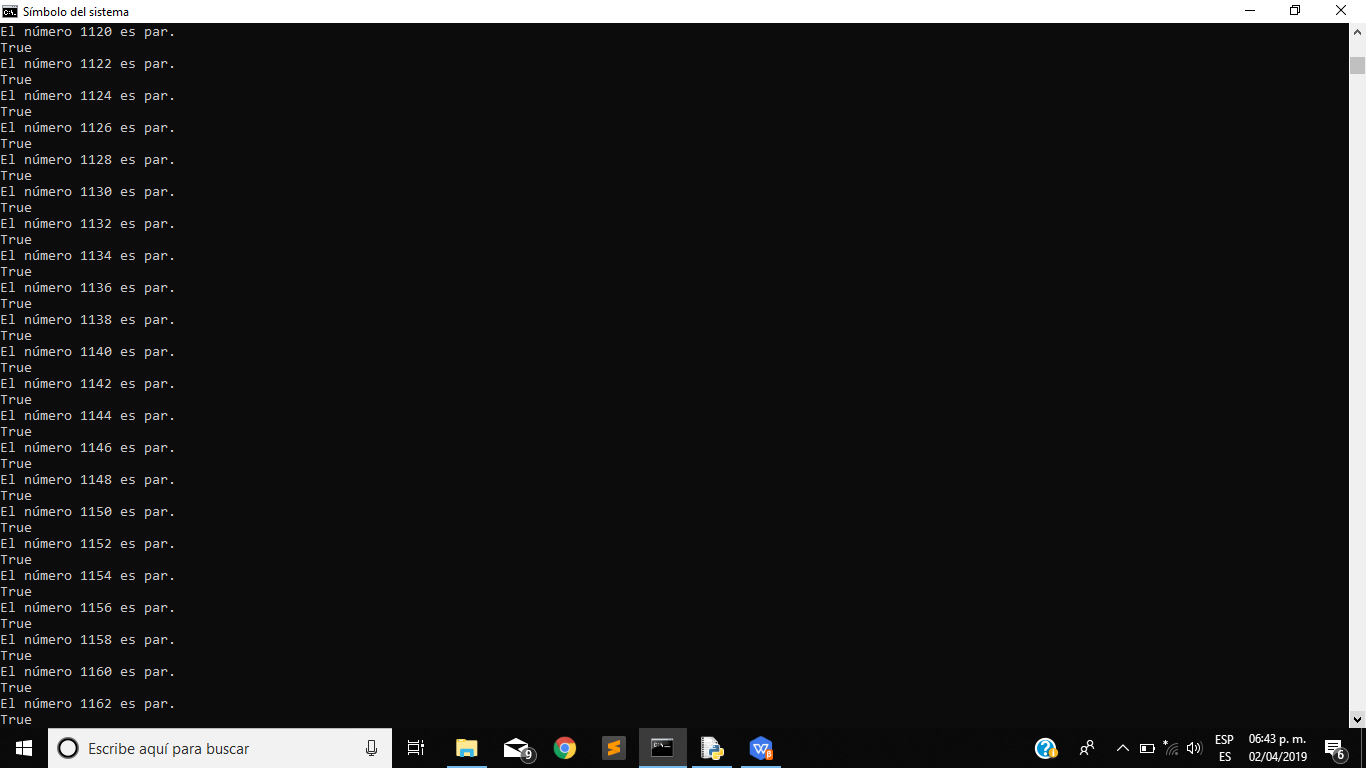
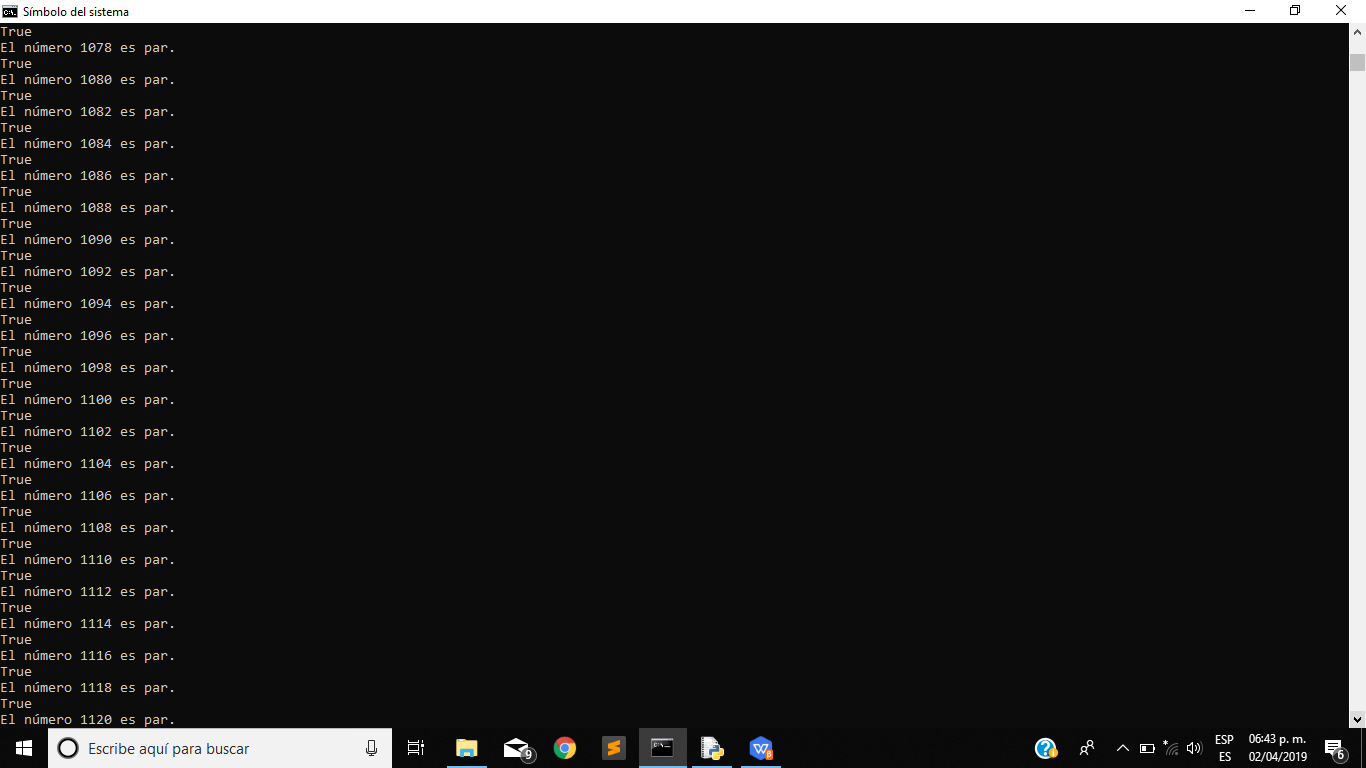
**Executing the coverage:**

Since it generates without any error, we can run the coverage to run the program and thus observe all the even numbers between the range that we mentioned in the main code (ejemplo.py), we use the command “coverage run ejemplo.py”.



Since we run it we will start to show all the even numbers between the selected range.



Already when it generates all the even numbers between the range of 1000 to 3000, and finally shows us the performance that the program had in a table, the cover was 73%.

