

**JSC “Kazakh British Technical University”**

**School of Mathematic and Cybernetics**

    Analysis of Data Bases

**Laboratory Work #10**

**(Python)**

**Prepared by: Maratuly Temirbolat**

**Almaty 2021**

**SHORT DESCRIPTION:**

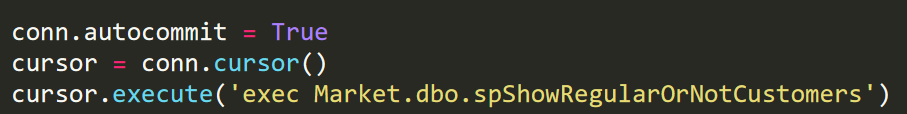
**The task of the Laboratory Work is to create queries using 5 Stored Procedures, 7 Triggers, 8 Functions. Then it is required to compile the queries using Python and show as well as write the results in console and file respectively.**

**The whole exercises were compiled in Sublime with Python (Version 3) extension. As long as we needed to use python we had to connect it to the sql server (especially to our server with the corresponded driver). The whole procedure is done here:**



**“import pyodbc” means that we import the already installed (manually) library in order to connect to our SQL Server (For MySQL we use another different library). Then we connect to server by ‘connect’ function where we fill the Driver, Server as well as connection by ourselves. Congratulations ! We have connected to our Database.**

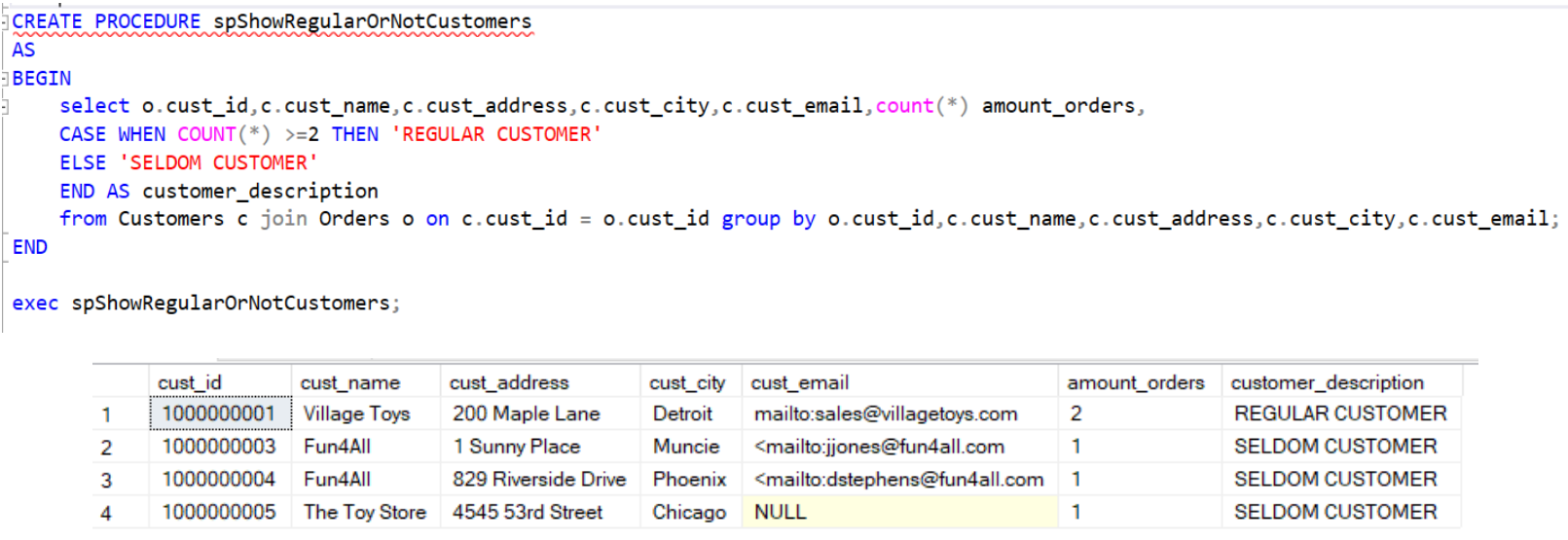
**The next step:**

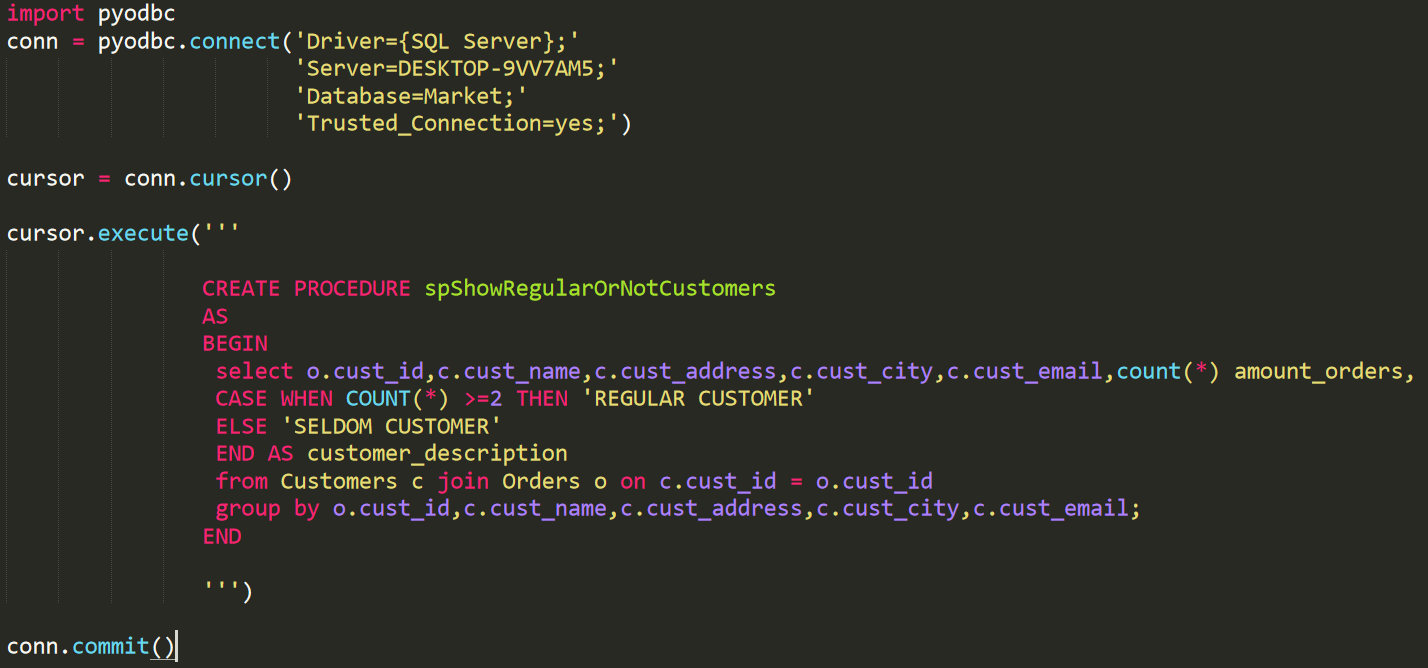


 **Here we also use internal variable ‘autocommit’ and assign the True value that means we want to commit all the queries automatically (without it won’t work). As we finished let’s keep moving on, we create variable, here ‘cursor’ and assign the value of the function ‘cursor’ which means if we change ‘cursor’ variable we change ‘conn’ and we can use the built in function by calling ‘cursor’ for the further work! It is very convenient than call each time ‘conn.cursor()’ instead of just ‘cursor’. The Last row is responsible for queries executions. We just type above created variable, here it is ‘cursor’ and use function ‘execute’ in brackets of which we need to write our query that we want to execute. Here it is exec spShowRegularOrNotCustomers. By the way, we have to use prefix ‘Name of the database’ and ‘dbo’ before the names of the tables , functions, procedures and so on since we did not type which database we want to use (I did not manage to handle this problem). Then the whole result of the query is written into cursor variable which we then just show or write into the file. However, there is one problem with type casting. So, let’s overcome it by using map casting it into the list! Then we just use loop ‘for’ to go through the list ‘var\_fix’ and then just show each variable under each index casting it into the string.**

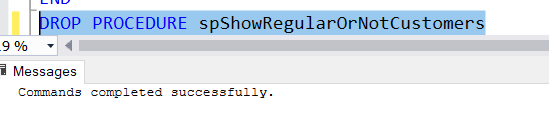
**Queries with 5 Procedures:**

1. Create a stored procedure that shows the customers and regularities of their visits to the market. The customer is said to be ‘REGULAR CUSTOMER’ if he/she bought at least 2 things, otherwise ‘SELDOM CUSTOMER’. Show all the info with this description.

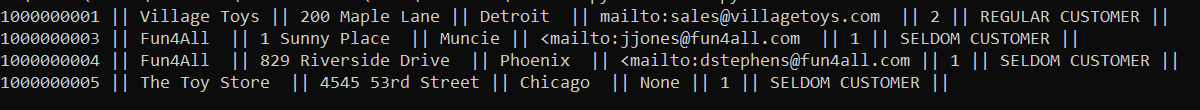
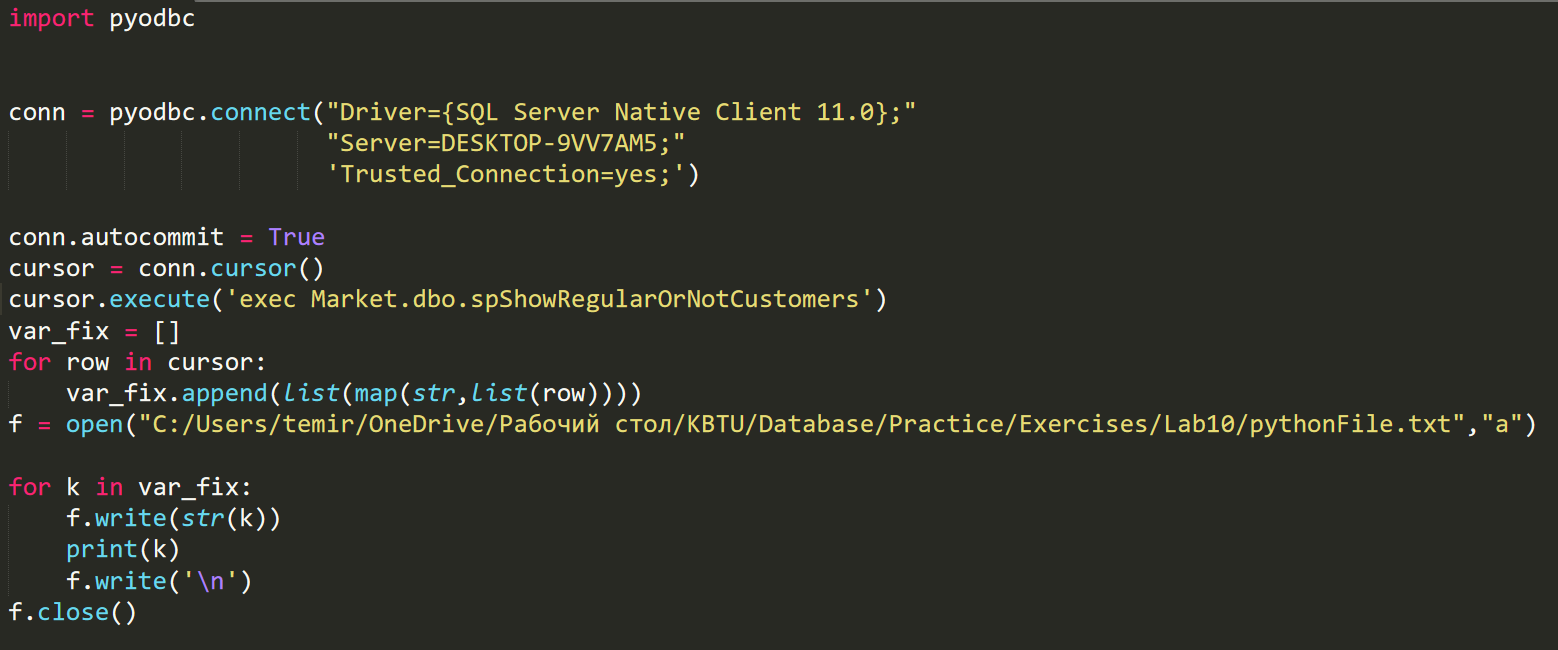
**SQL code with expected and needed output:**

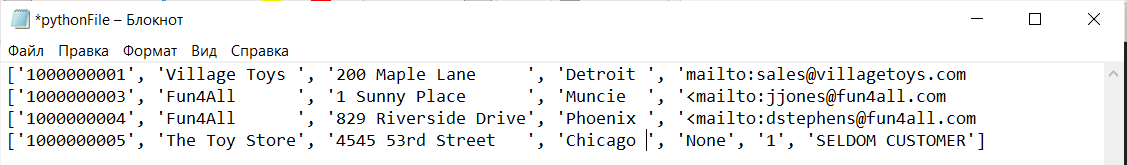
**The python code for creation the Procedure:**

**Be aware that we have already deleted the Procedure in advance:**

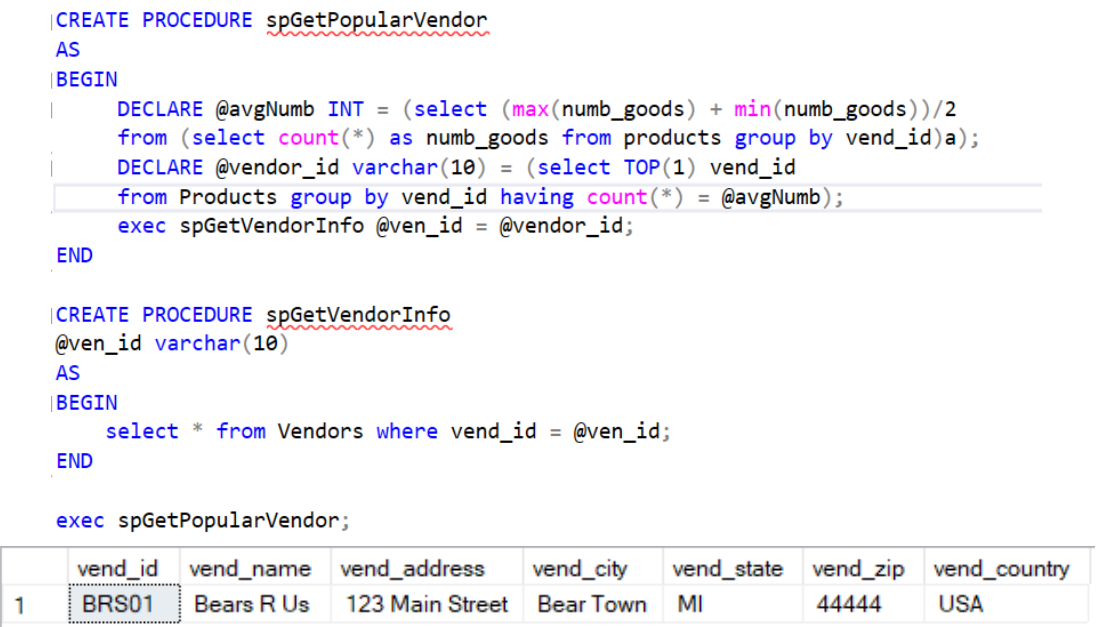


**The python code with appropriate output in console and in txt file which is also created during the compilation of the Python code:**

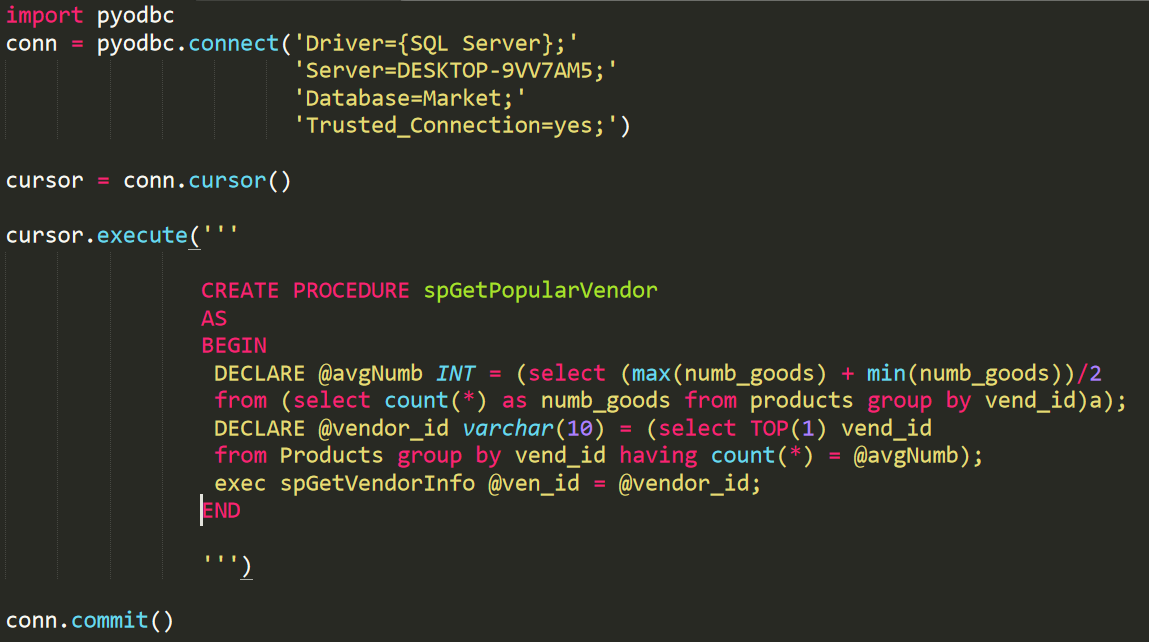
**The result in console which is the same but not beautiful****:**

**The same result in txt file although looks more structured**

1. Investigate a stored procedure that illustrates the most popular seller among all of the them. Use two procedures if it is necessary then show all the info about this vendor

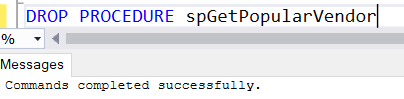
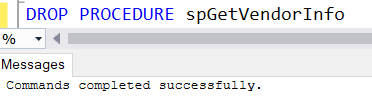
**SQL code with expected and needed output:**

**The python code to create the procedures:**



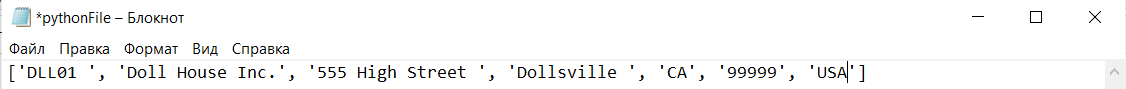


**Be aware that we have already deleted the Procedures in advance:**

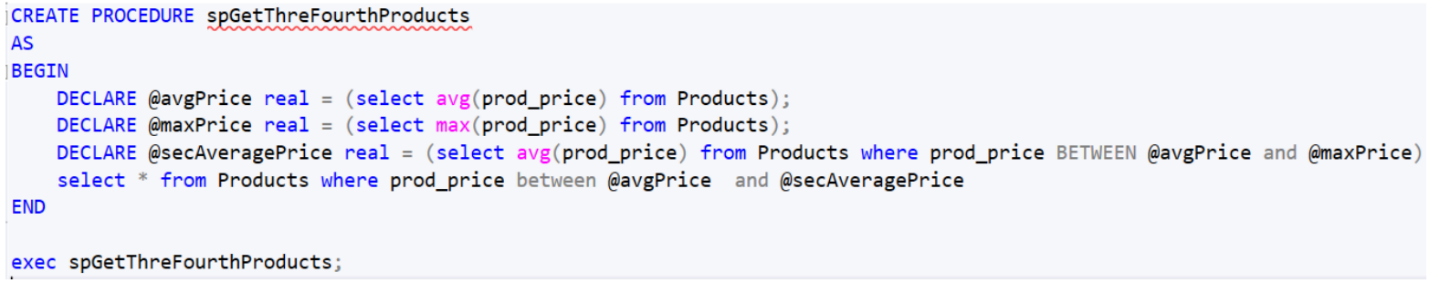
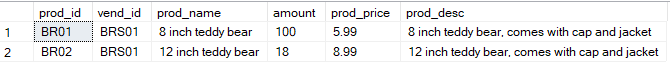


**The python code with appropriate output in console and in txt file which is also created during the compilation of the Python code:**

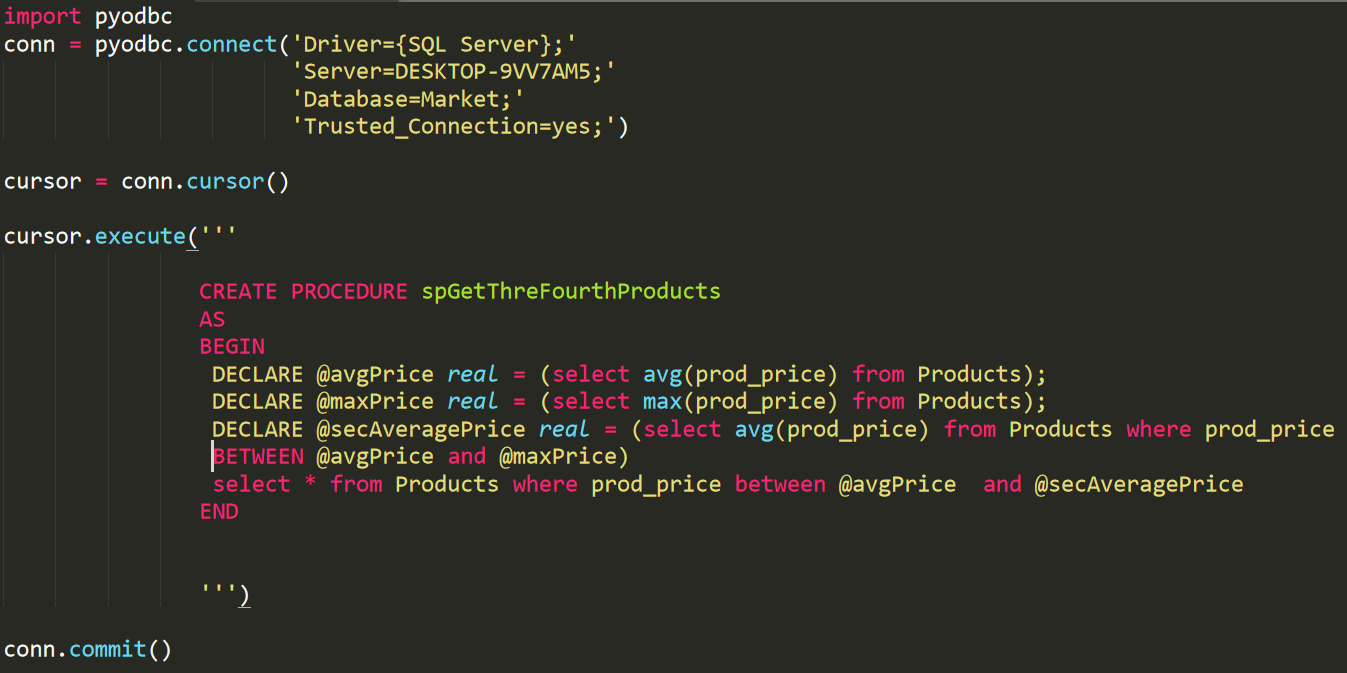
**The result in console:**

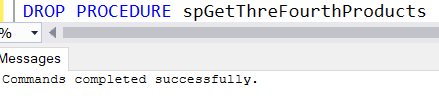
**The result in txt file:**

1. Provide a stored procedure that shows all the products whoes price is located between average price and second higher average price of the products. The second higher average price is located exactly in the middle of the mean value and max value price of the products.

**SQL code with expected and needed output:**

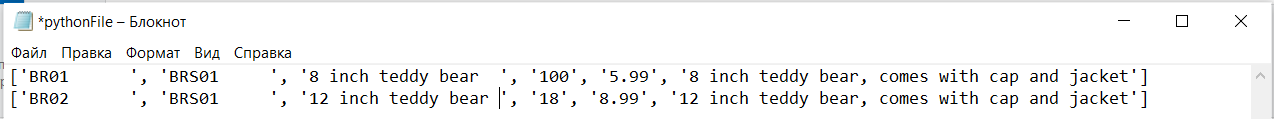
**The Python code for creation stored Procedure:**

**Be aware that we have already deleted the Procedure in advance:**



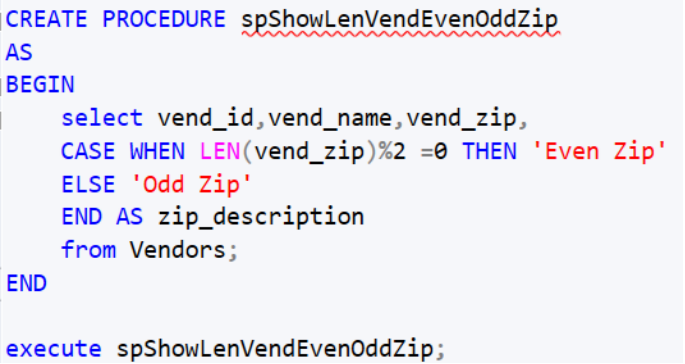
**The python code with appropriate output in console and in txt file which is also created during the compilation of the Python code:**

**The Console output:**

 **The file txt output:**

1. Create a stored procedure that shows all the id, name, zip as well as description of all the vendors. The description must indicate whether or not the length of the zip ODD or Even.

**SQL code with expected and needed output:**

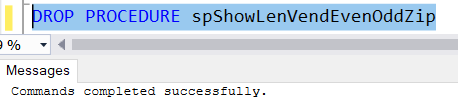




**The Python code for creation:**



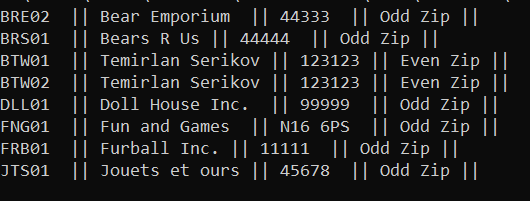
**Be sure that we deleted the Procedure in advance:**

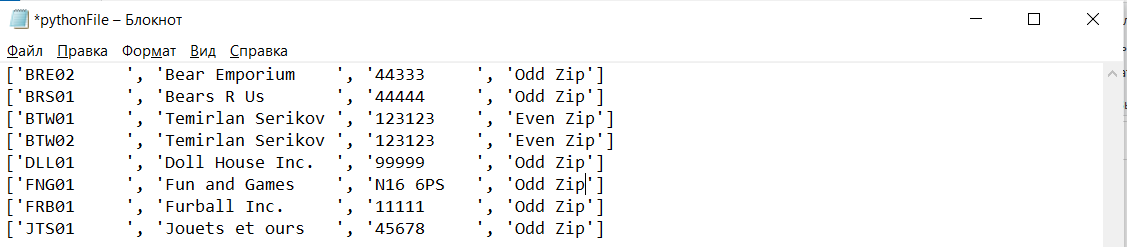


**The Python code:**

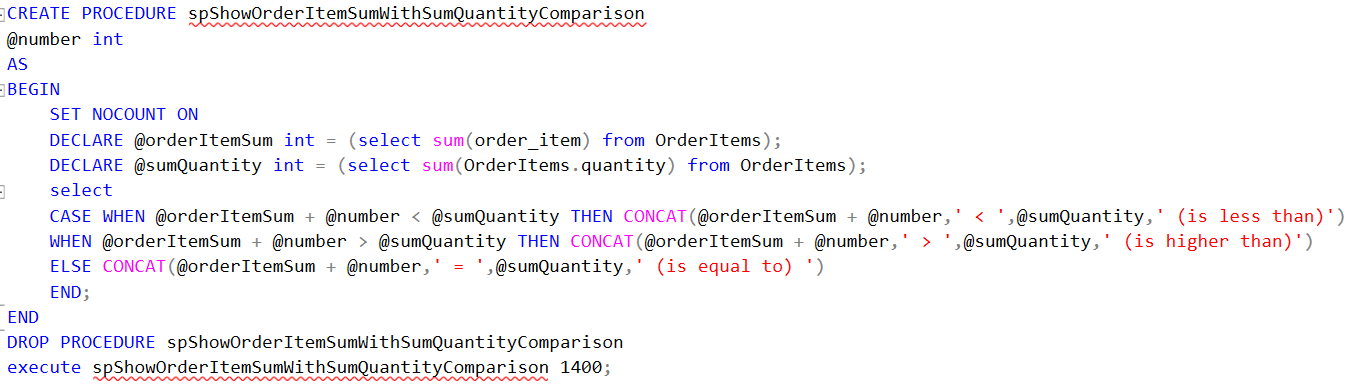


**The console output:**



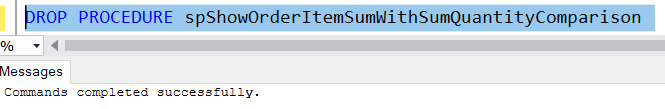
**The txt file output:**

1. Create a stored procedure which takes any integer number and selects the sum of the whole items + given number less,equal or higher than the total sum of the whole quantities of the items. See solution below:

**SQL part with expected and required output:**



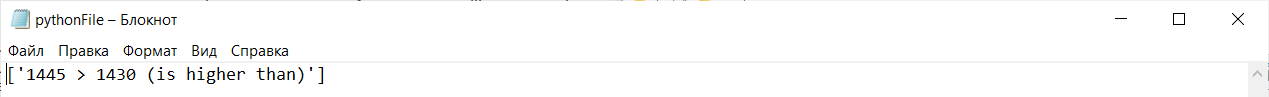
**The python code for creation PROCEDURE:**

**Be aware that we deleted the PROCEDURE IN ADVANCE:**

 **The python code to execute the procedure:**

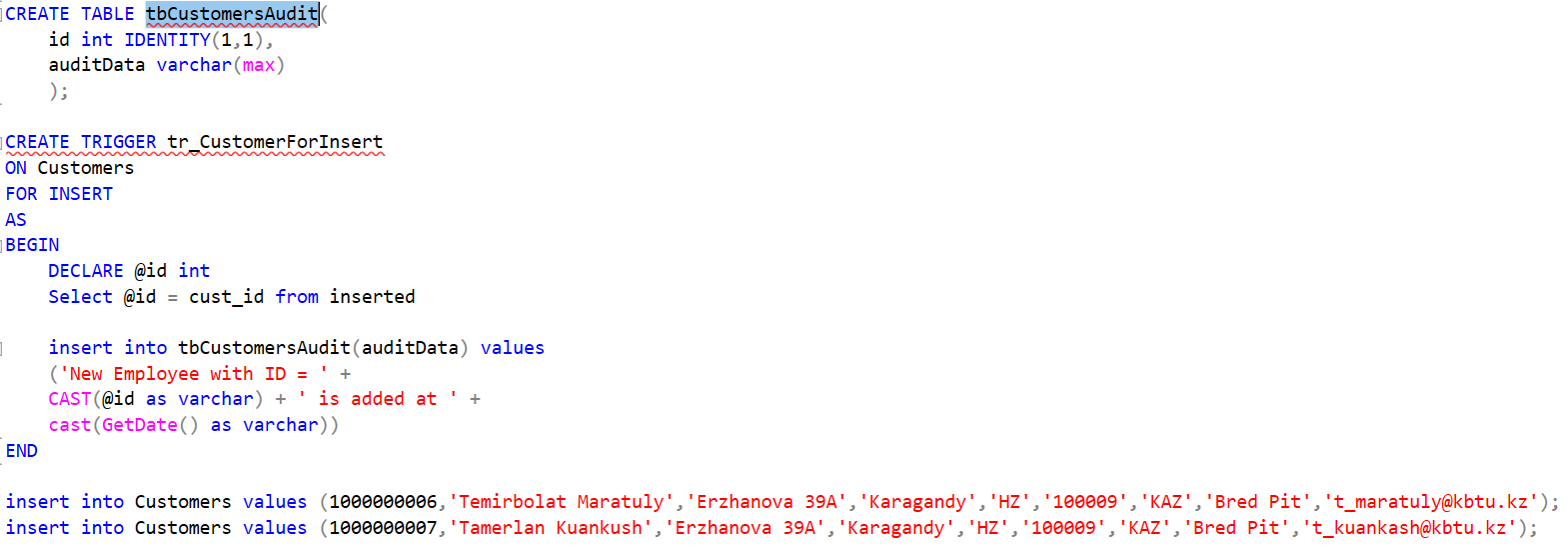
**The result in console:**



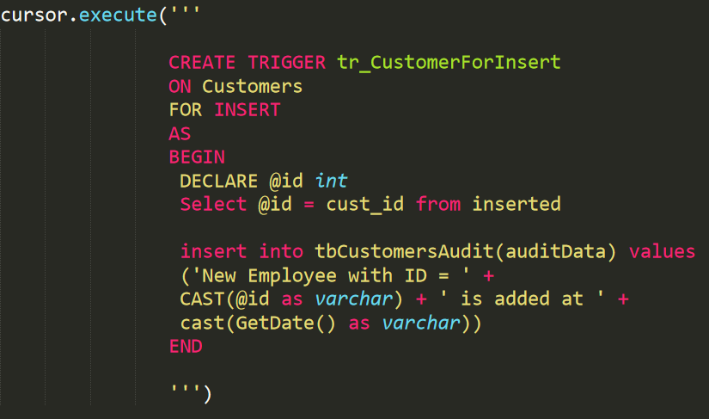
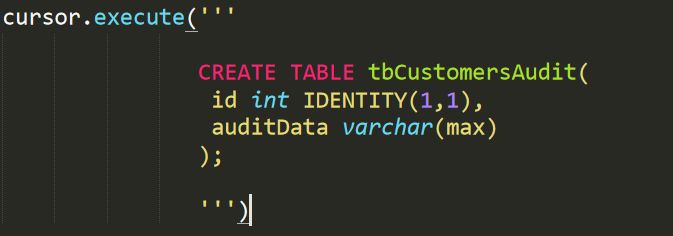
 **The result in txt file:**

**7 queries for Triggers (Very Interesting)**

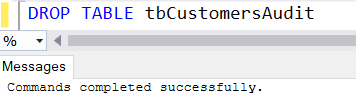
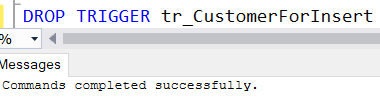
1. Create a new table and call it tbCustomersAudit with 2 columns: id int with identity starting from 1 as well having step 1 and auditData varchar(max). As you finished, create a trigger that would add the information about Customer who was inserted into the Customers table with his/her id and time when he was added.

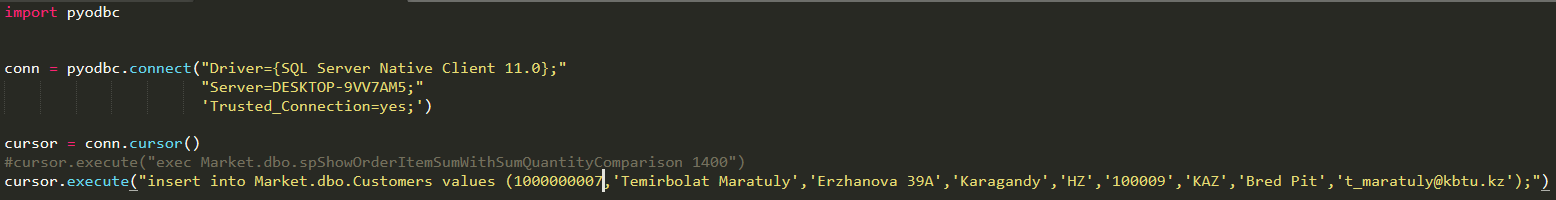
**SQL code with appropriate output:**



 **The Python code for creation TABLE and Trigger:**

**Be aware that we have already deleted TABLE as well as Trigger in advance:**



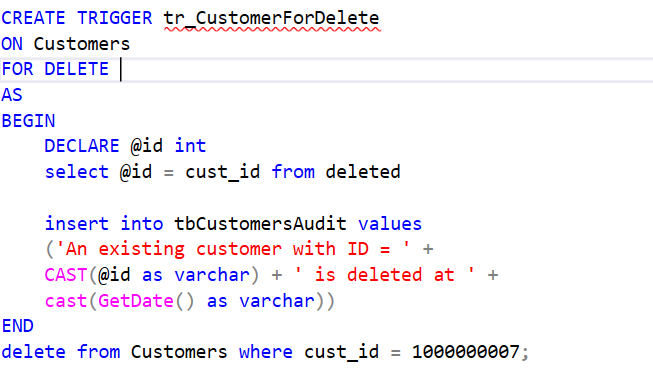
 **The Python code to insert data:**

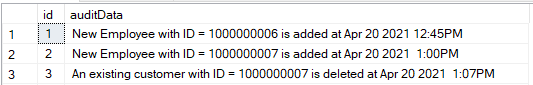
 **The Python code to see Audit:**

 **The code output:**

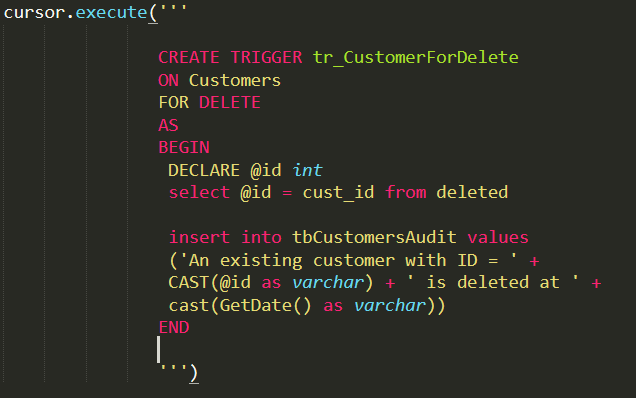
1. Do about the same procedure as in 6-th exercise. However, you have to do it with DELETE Part and write the information down about activities into recently created table. Type “An existing customer with ID = … is deleted at … (TODAYS DATE)”

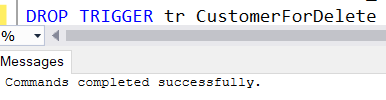
**SQL code with appropriate output:**





**The Python code to create the trigger:**

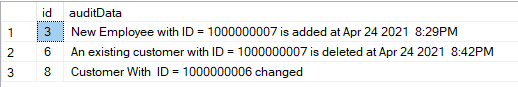


 **Be aware that we had already removed Trigger in advance:**

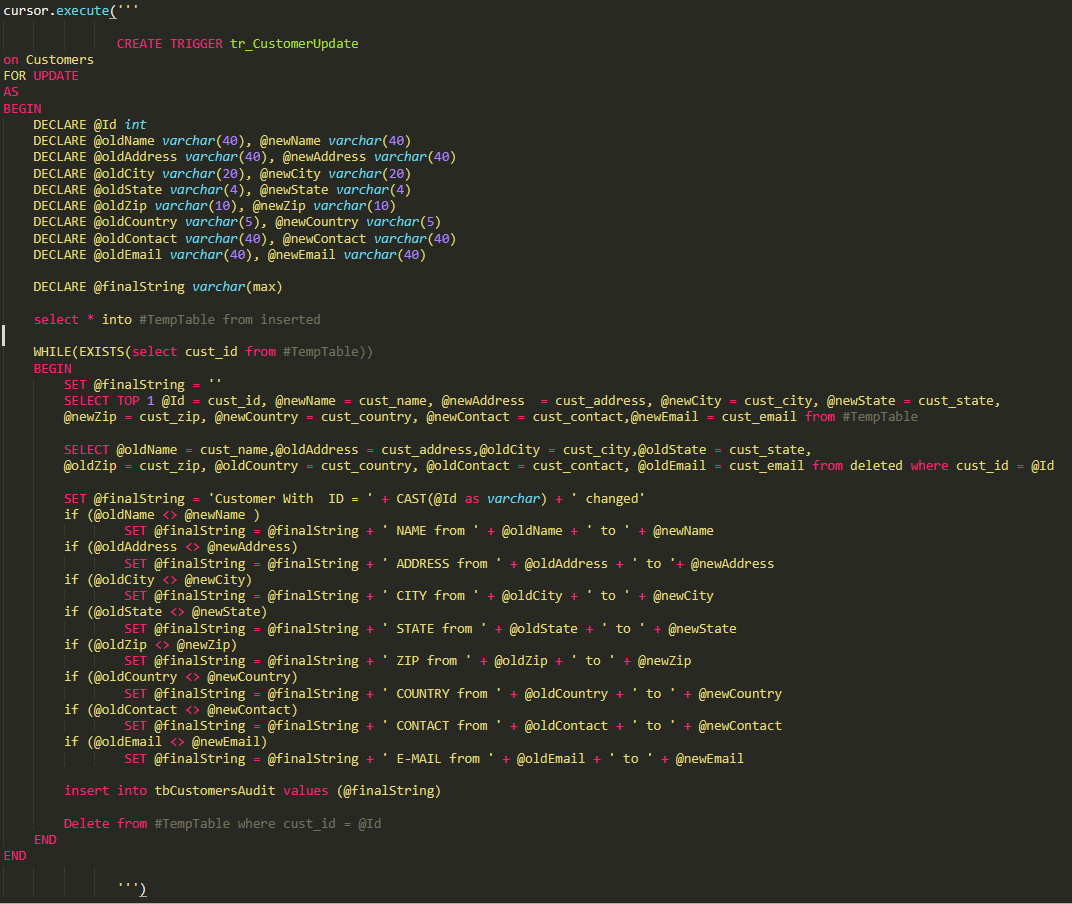
 **The Python code to show the working principle:**

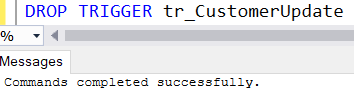
 **The console output:**

1. Again. Create a Trigger that is responsible for indicating the information that customers changed about themselves and add this notification into the table that was create in 6-th exercise. The information would be too long because of the number of given attributes in the CUSTOMERS table.

**The SQL code with corresponded output:**

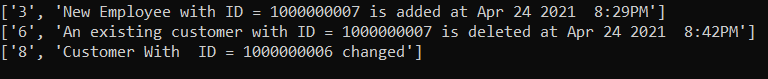
**The Python code to create the Trigger:**

**Be sure That we had deleted the Trigger:**



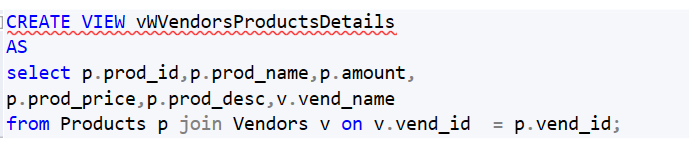
**The Python code to see the working principle of Trigger:**

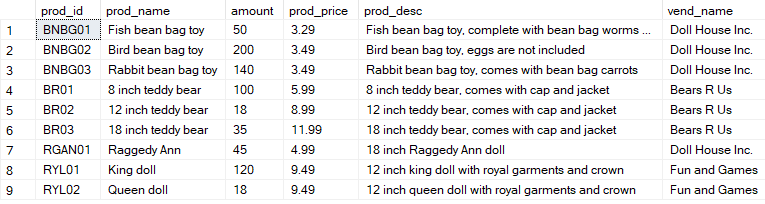


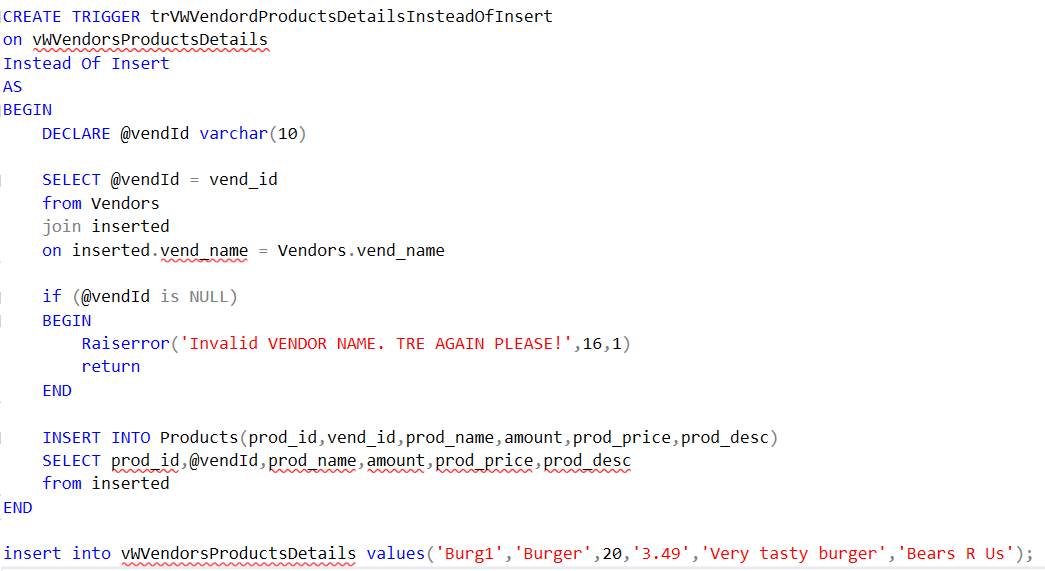
 **The Python output in console:**

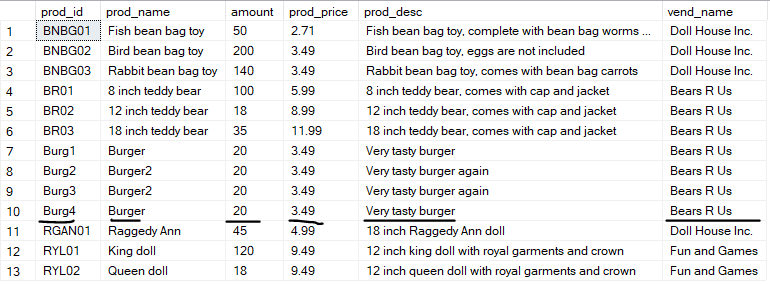
1. Wow, the previous task was quite unexpected but what if we additionally create a VIEW (virtual table) which contains the join(combination) of products and vendors. We need to use this combination as one table make some manipulations for it. If we write insert into nameOfTheView values we would obtain a mistake because it consists of the multiple tables. So, for this purpose we use TRIGGER. However, we create it with INSTEAD of Insert to imitate the insert procedure.

**The SQL code with corresponded output:**

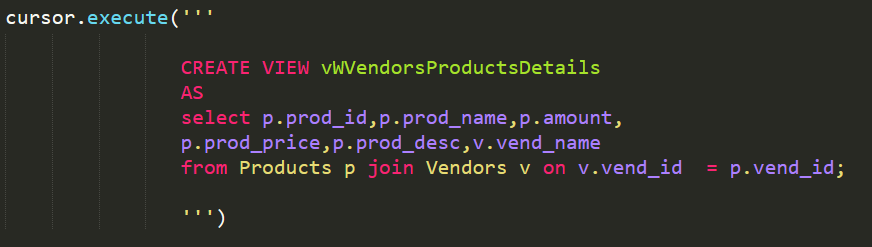
CREATED VIEW:

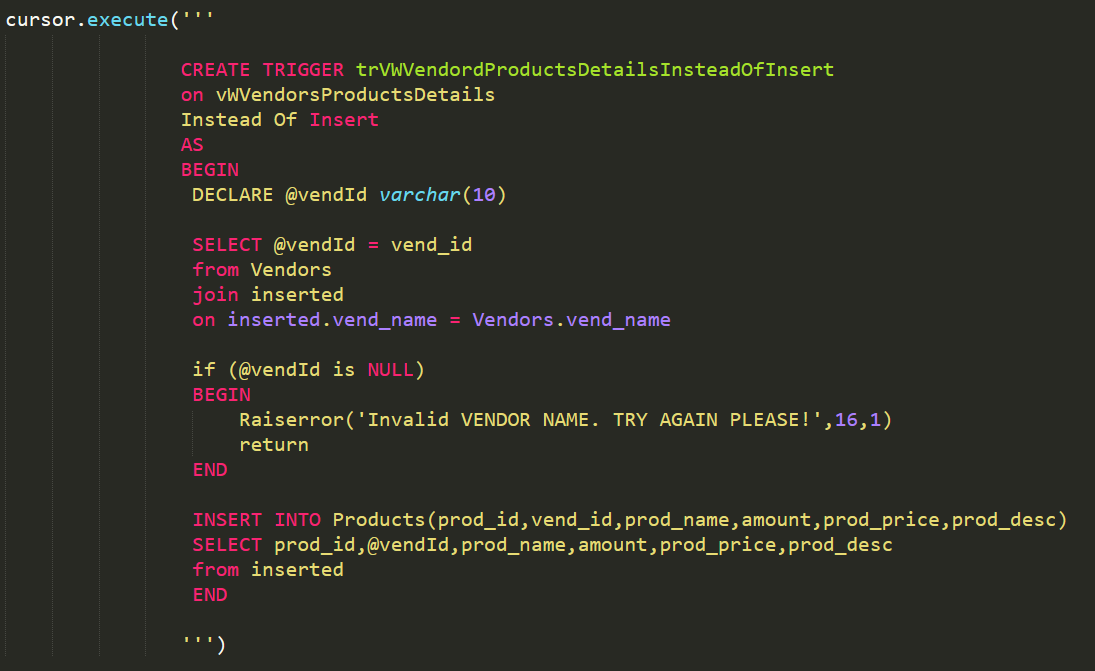


Create a TRIGGER FOR INSERT:

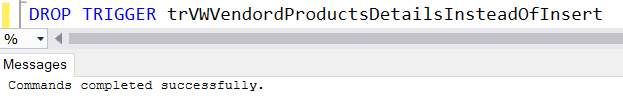
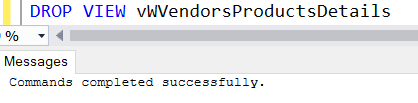


**The Python code to create View and Trigger:**

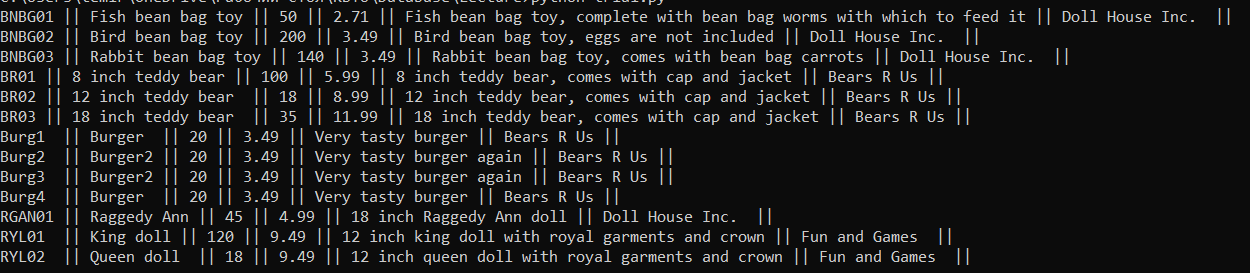




**Be aware that we deleted Trigger and View in advance:**

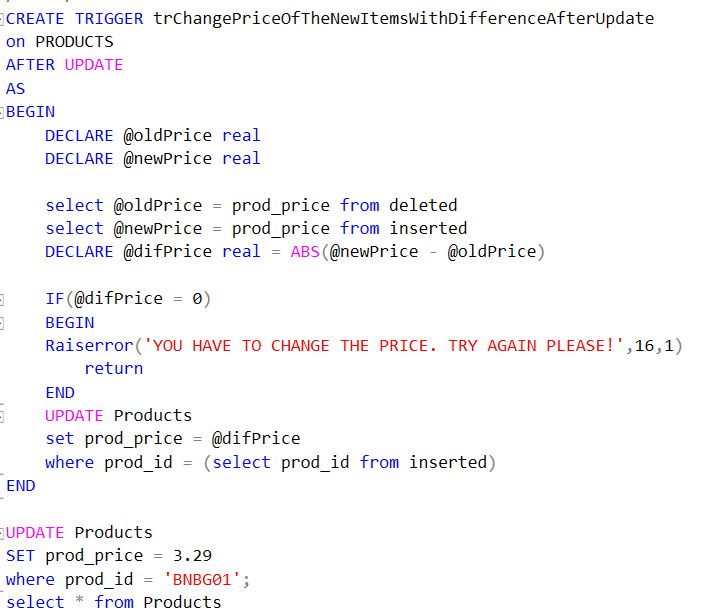


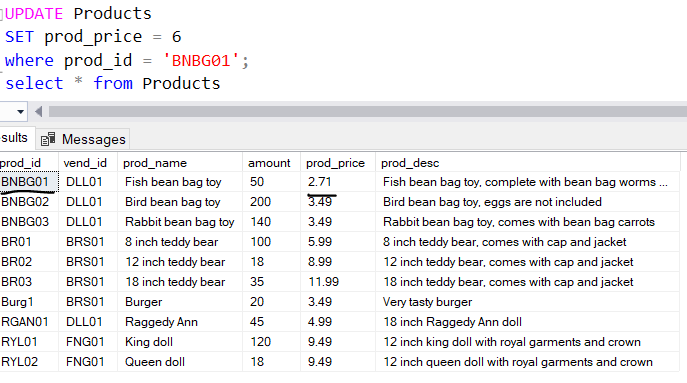
 **The Python code to execute and see the result:**

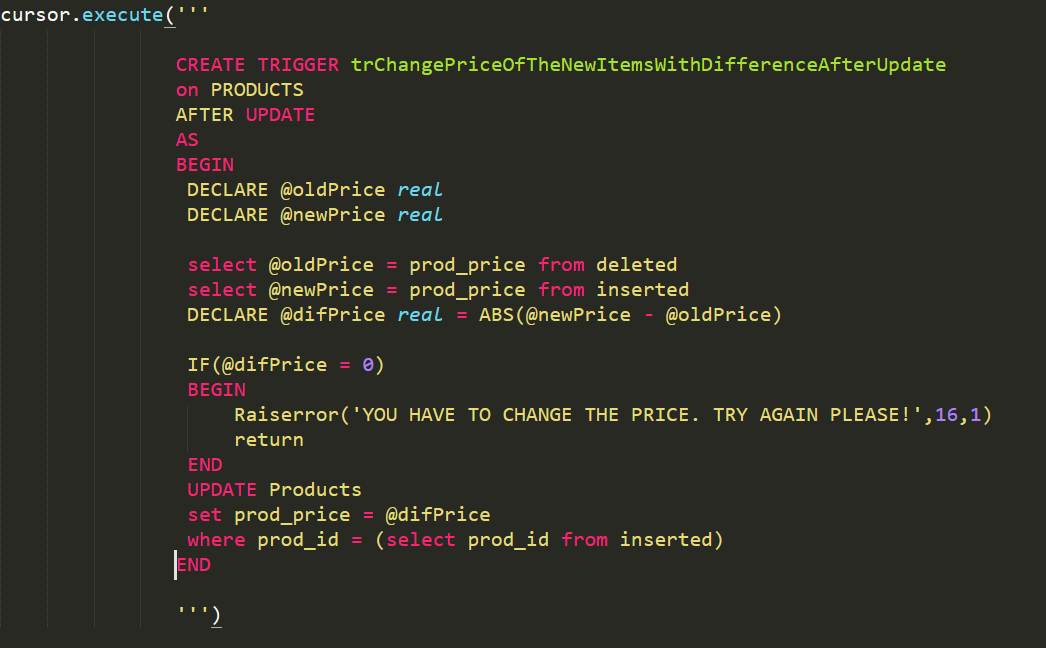
 **The result in Console:**

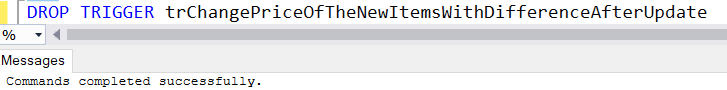
1. Create a Trigger that works if we update the PRICE of the PRODUCTS otherwise mistake appears. We need to assign the difference between new and old prices as new price for those products that were influenced by changes.

**The SQL code with output:**

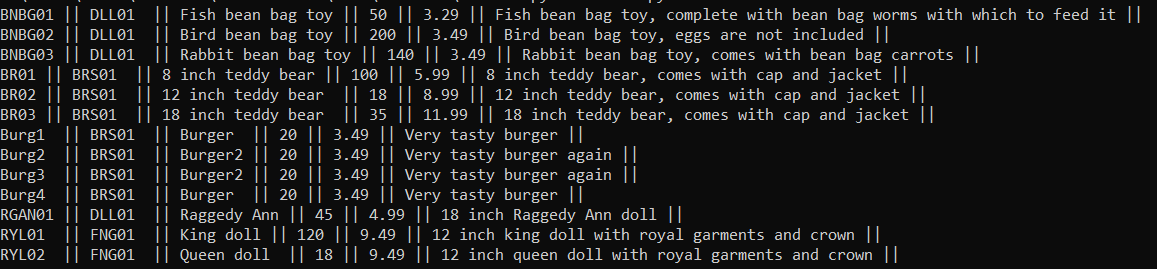




 **The Python code to create Trigger:**

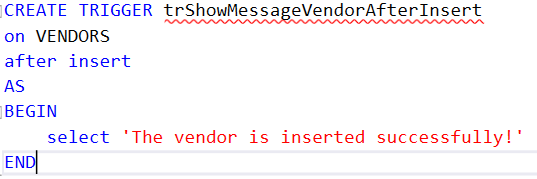
 **Be confident that deleted the trigger before:**

 **Execute the working code:**

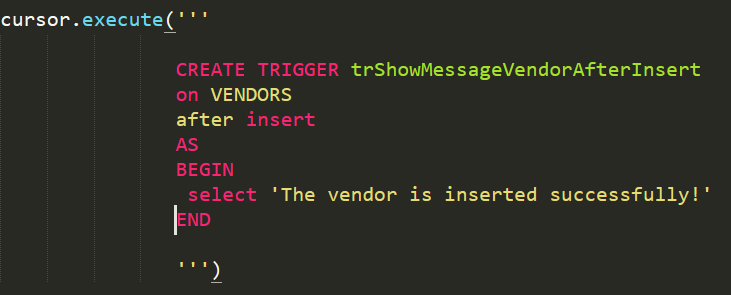
 **The output in console:**

1. Oh, since there are no efforts and imagination to produce a cool query let’s create something basic. Write a Trigger that would show ‘The vendor is inserted successfully!’ if there was inserted a new Seller into the table.

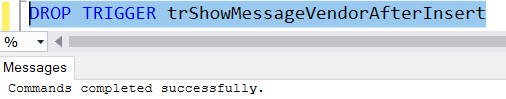
**The SQL code with corresponded output:**





 **The Python to create Trigger:**

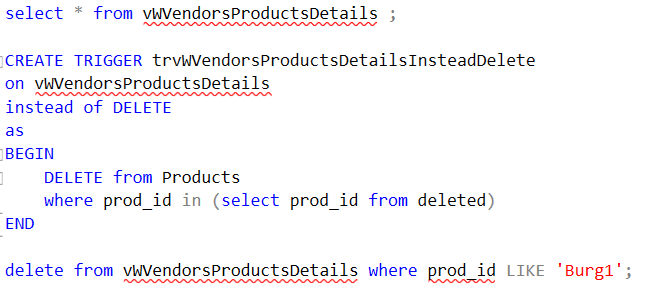
**Be sure that we deleted Trigger before:**

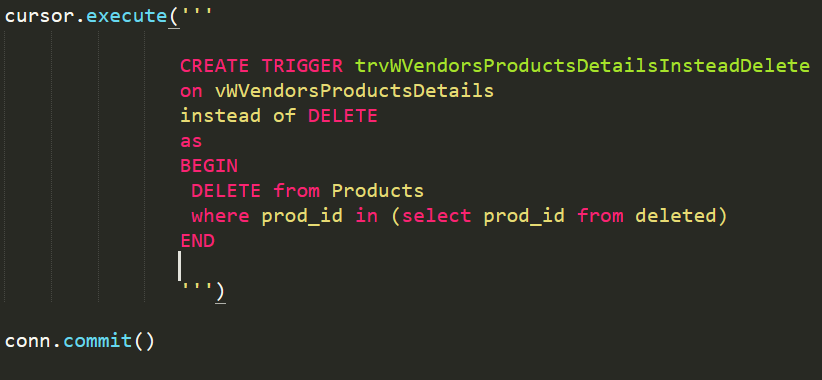


 **The python code to execute the Trigger:**

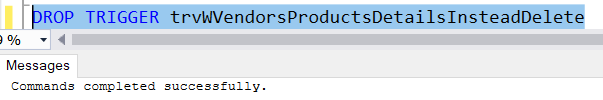
 **The Output:**

1. Well, you need to use the created previously VIEW and this time investigate a TRIGGER that would imitate the working principle of DELETE for the VIEW since mistakes appears if we try to do it. We will delete recently inserted burger product.



  **The creation in Python:**

**Be confident that we deleted the Trigger in SQL before:**

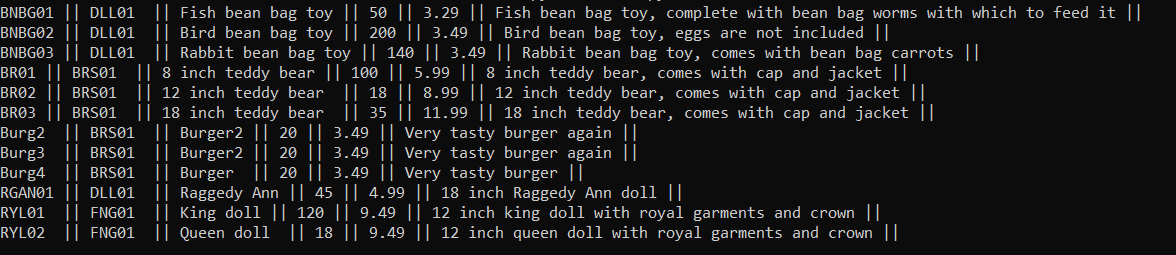


**The Python code execution:**



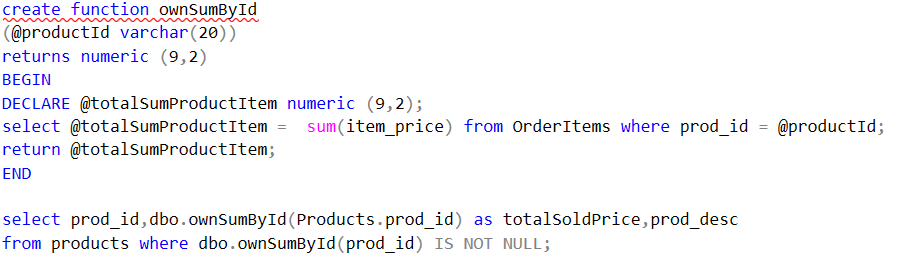
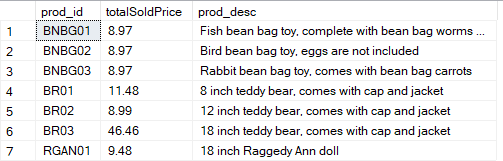


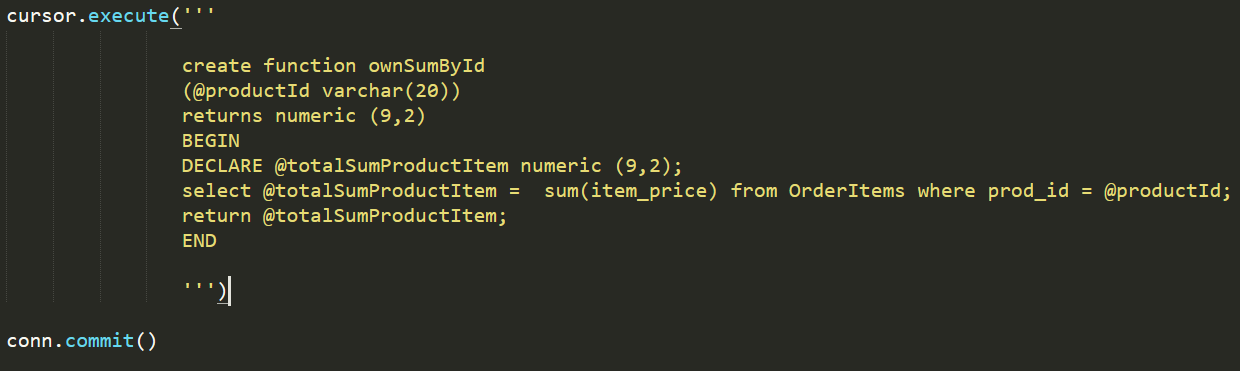
**The output with no ‘Burg1’:**



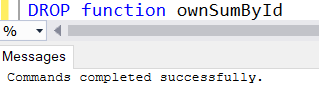
**8 QUERIES FOR FUNCTIONS  
 (Also very interesting)**

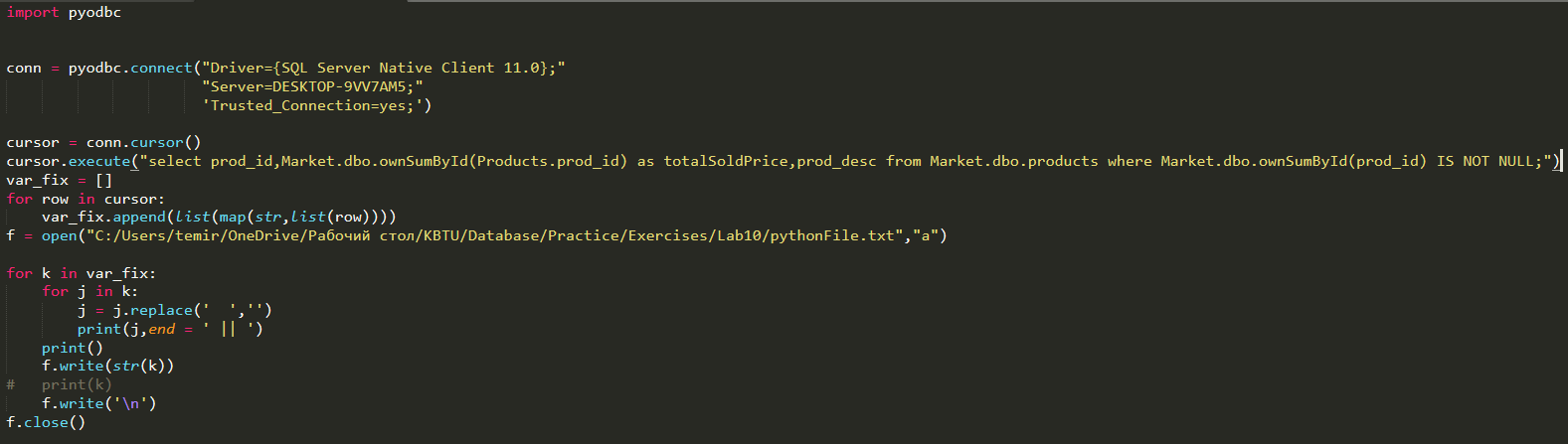
1. Create a function that would replace the real sum for the prices of the products and return this value.

**The SQL code with appropriate output:**

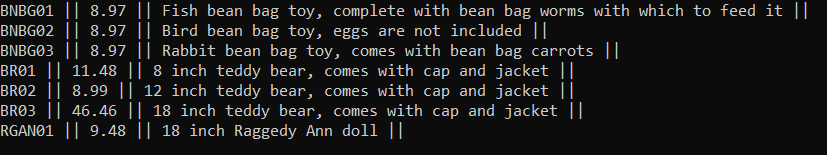
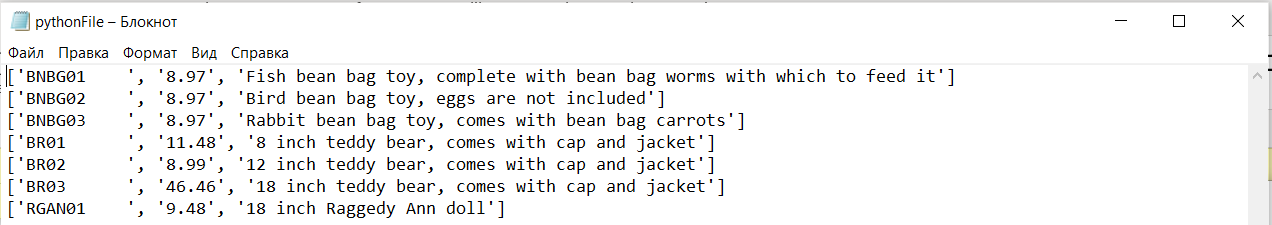
**The Python Code to create the Function:**

**Be sure that we deleted the function previously:**

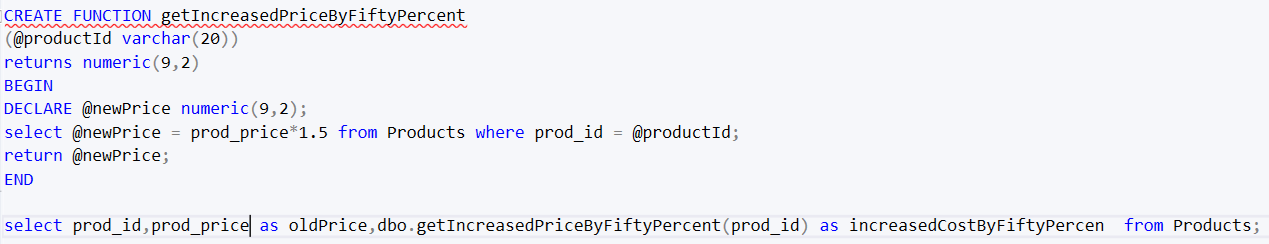
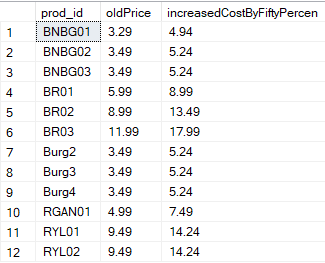


**The Python code to execute the Function and get the result:**

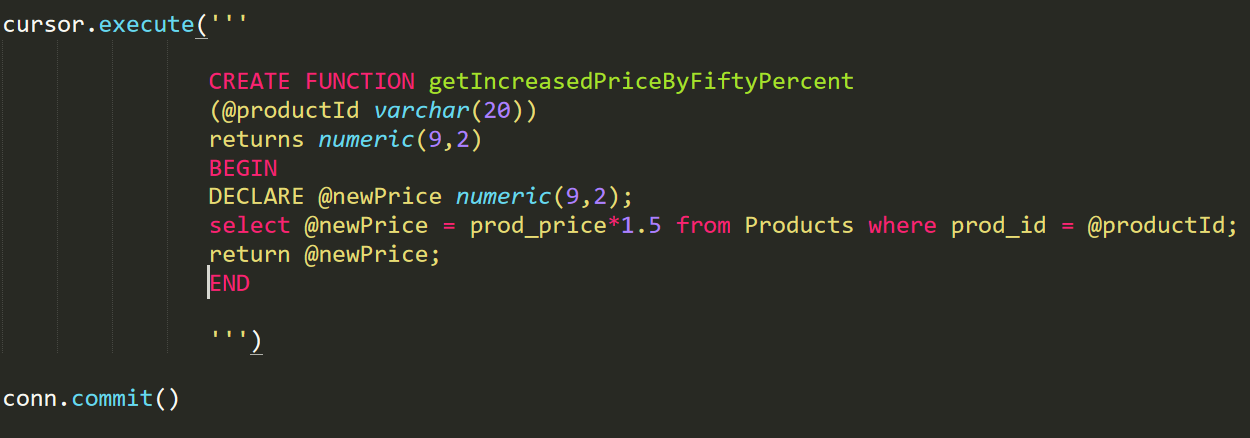
**The output in console:**

**The output in txt file:**

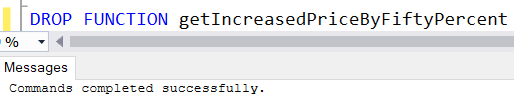
1. Write the function that would increase a price of the products by 50 percent and returns this new price.

**The SQL Part with code and output:**

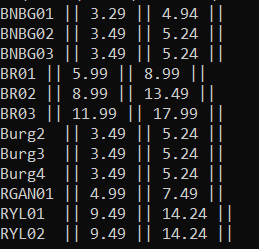
**The Python code to create the Function:**

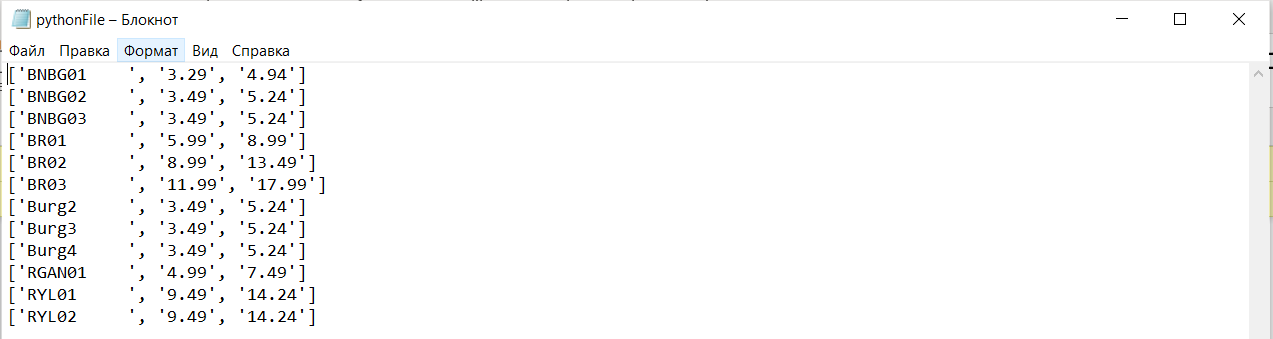


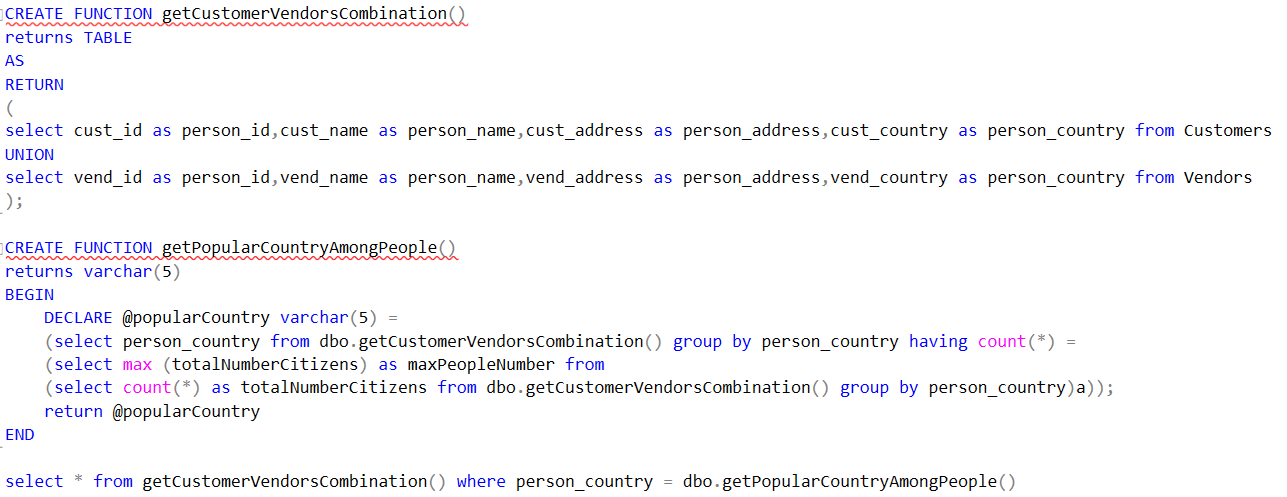
**Be aware that we deleted the function in advance:**



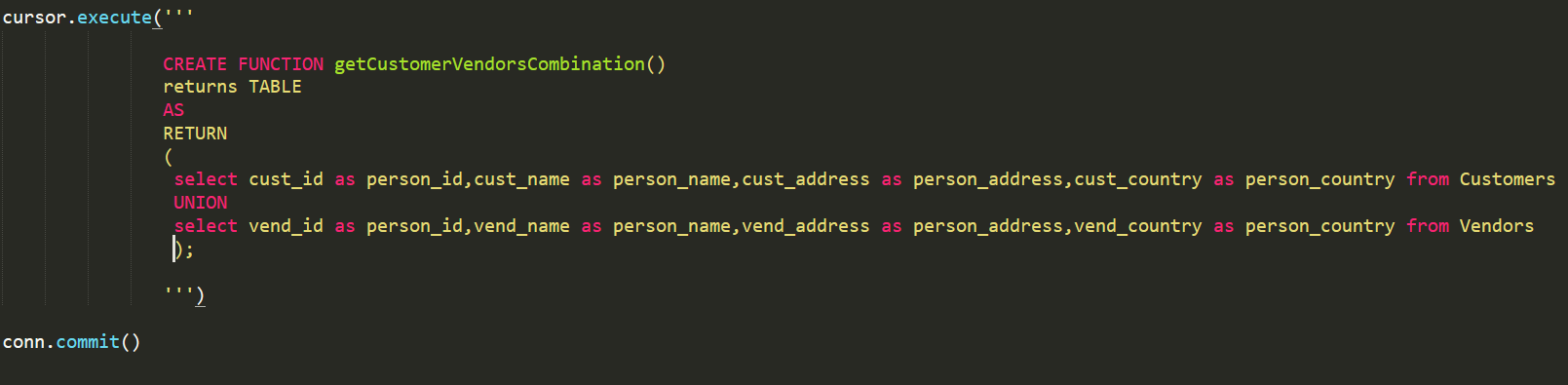
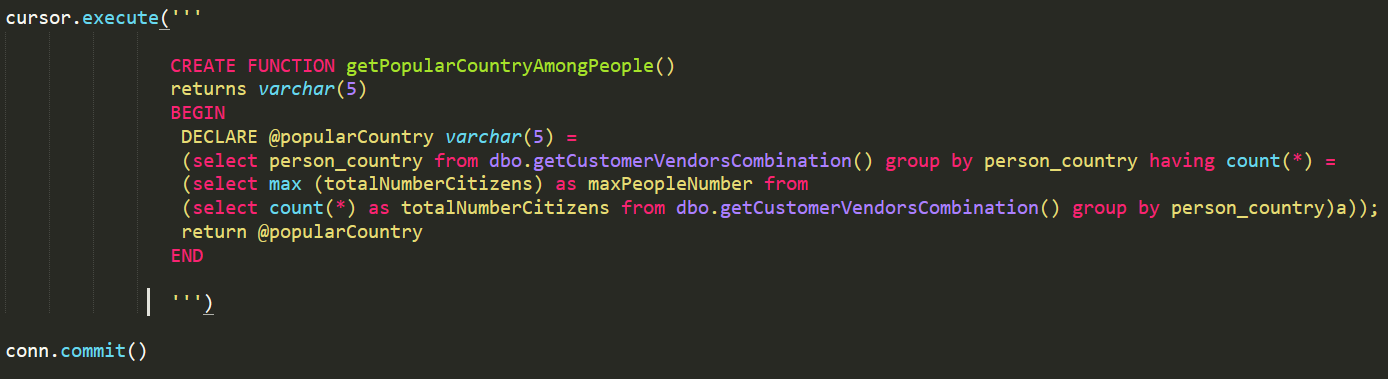
 **Python file with Function Execution:**

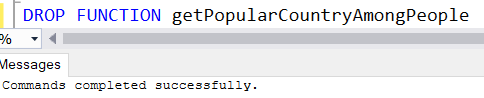
 **The Python output in console:**

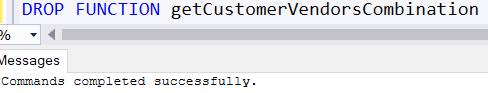
**The output in txt File:**

1. Write to functions where the first is responsible for the union of customers and vendors and it is returned as a table while the second function needs to take this new table from the first function. The second function has to return the most popular country among people. Show the people who live in the most popular Country.



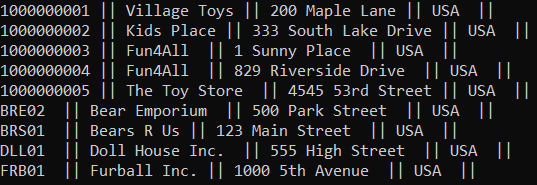
**The Python Functions Creation:**

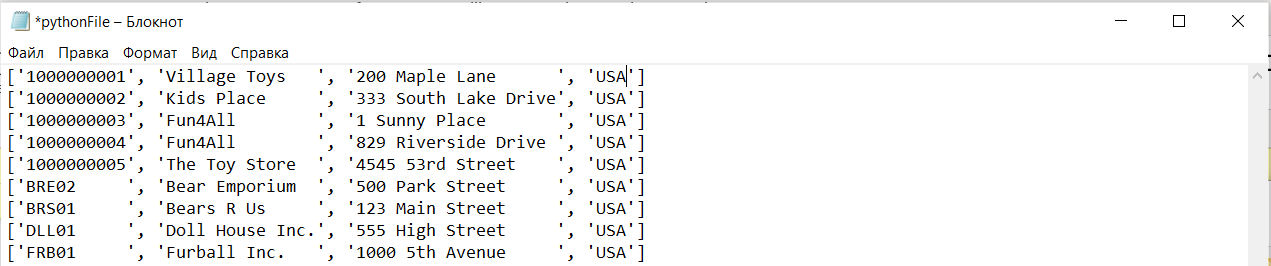
 **Be aware that we deleted the functions:**



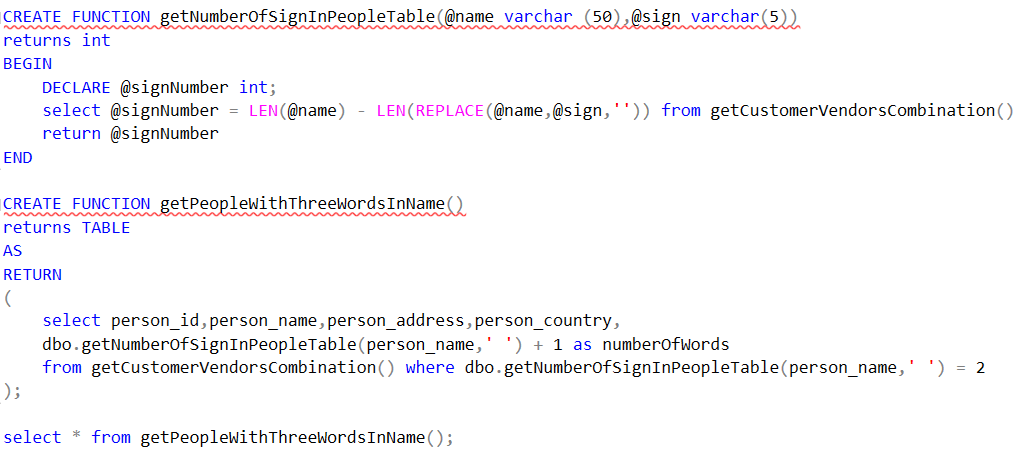
**The Python File to execute the functions:**

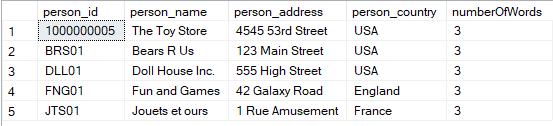
**The output in the Console:**



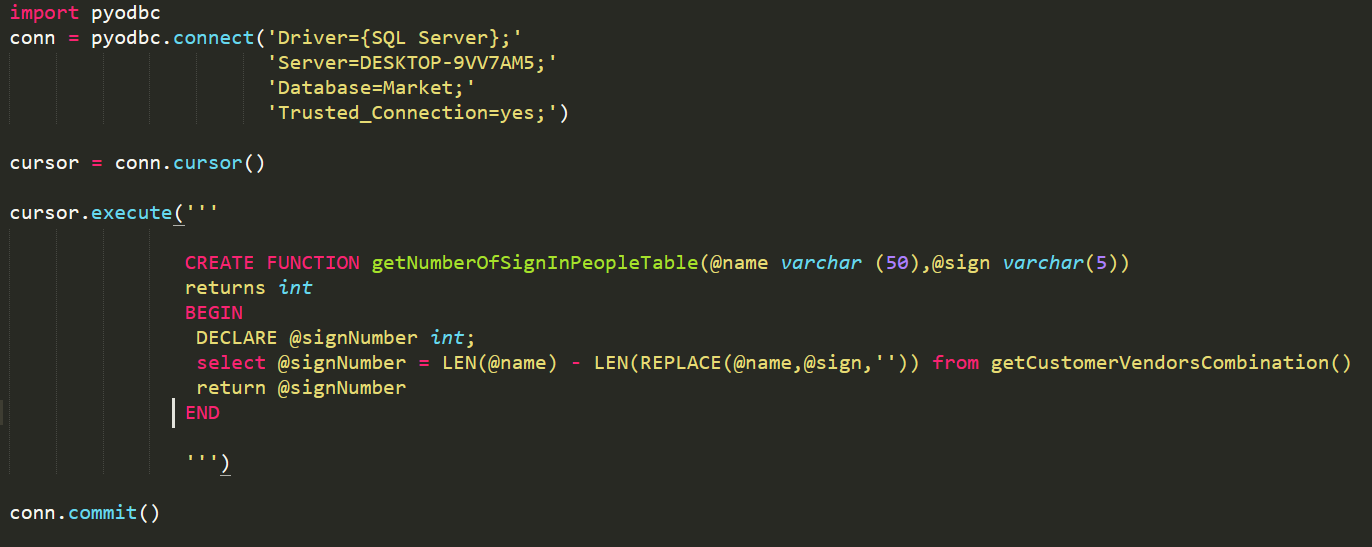
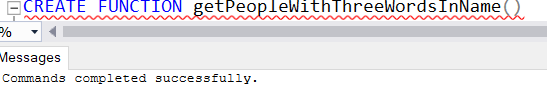
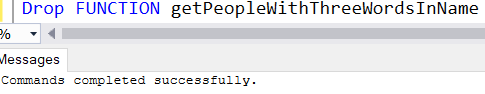
 **The Output in the txt File:**

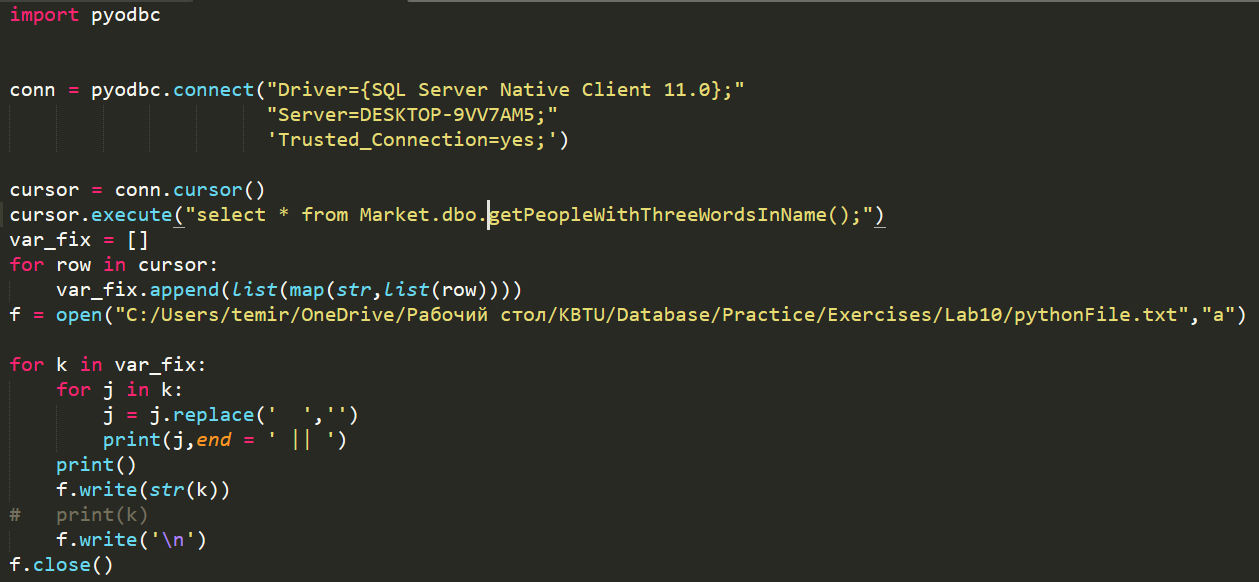
1. Write the functions that finally return a table with those people whose names consists of 3 words. Then show the whole information of these people.

**The SQL Code with corresponded output:**

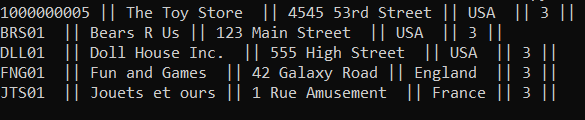


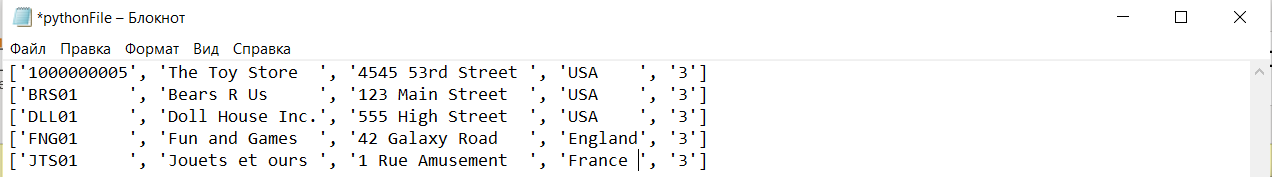
**The Python Creation of Functions:**

 **Be sure that we deleted the Function in advance:**

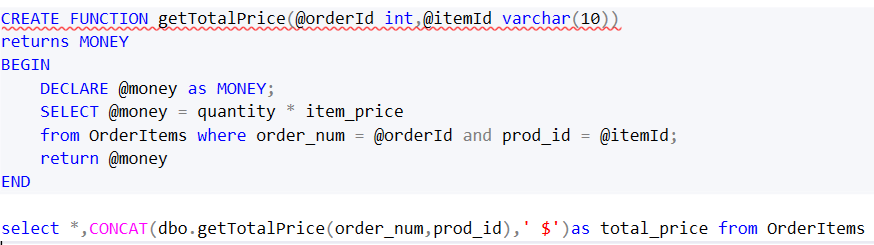
 **The Python code with Functions Execution:**

**The output of the code in Console:**

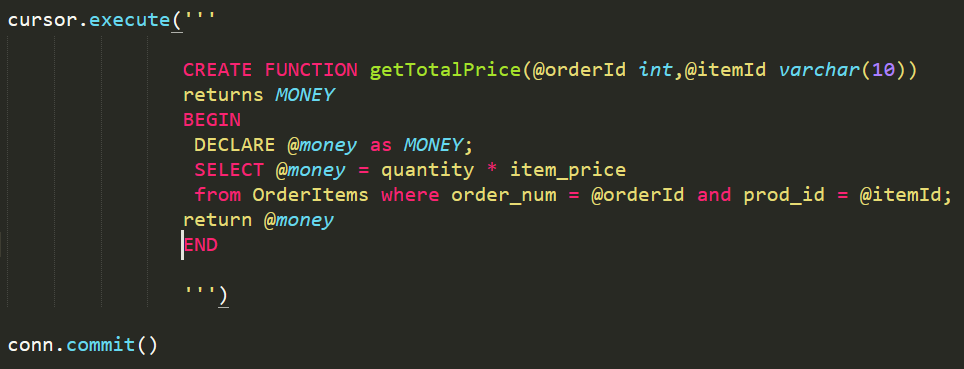


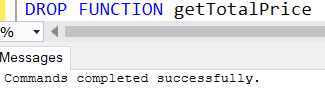
 **The output in txt File:**

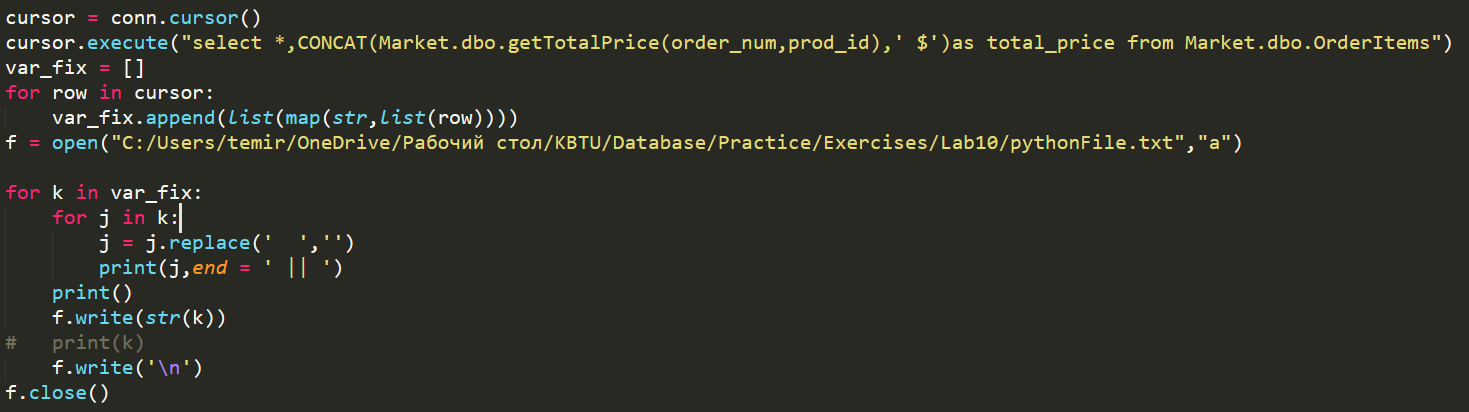
1. Write a function that would return the total price that person needs to pay. Just multiply the quantity of items by cost of one. Use the type MONEY with dollars sign to make it more realistic.

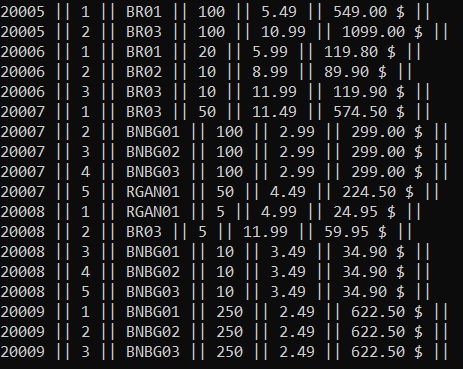
**The SQL code with needed output:**

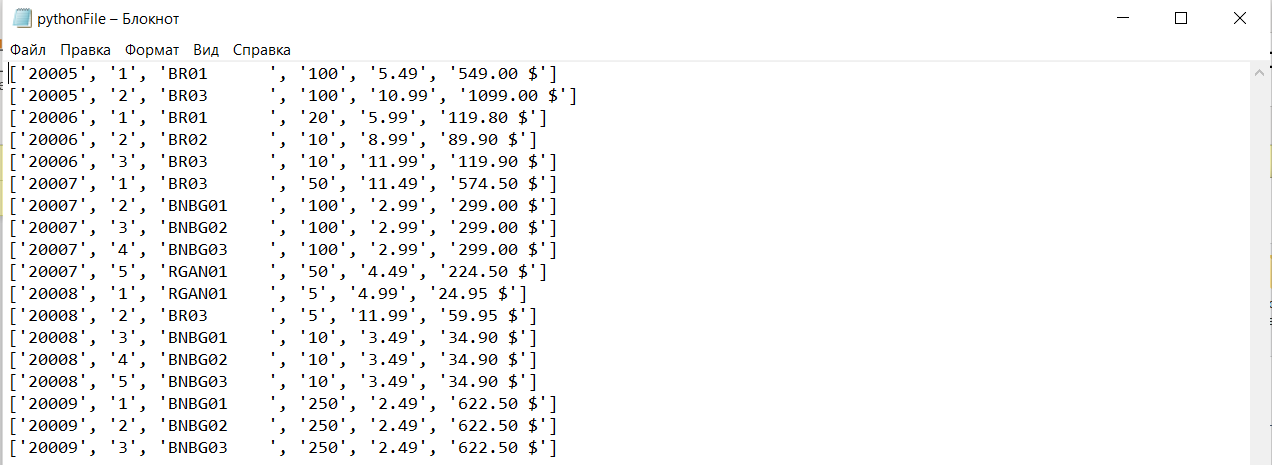


 **The Python Creation of the Function:**

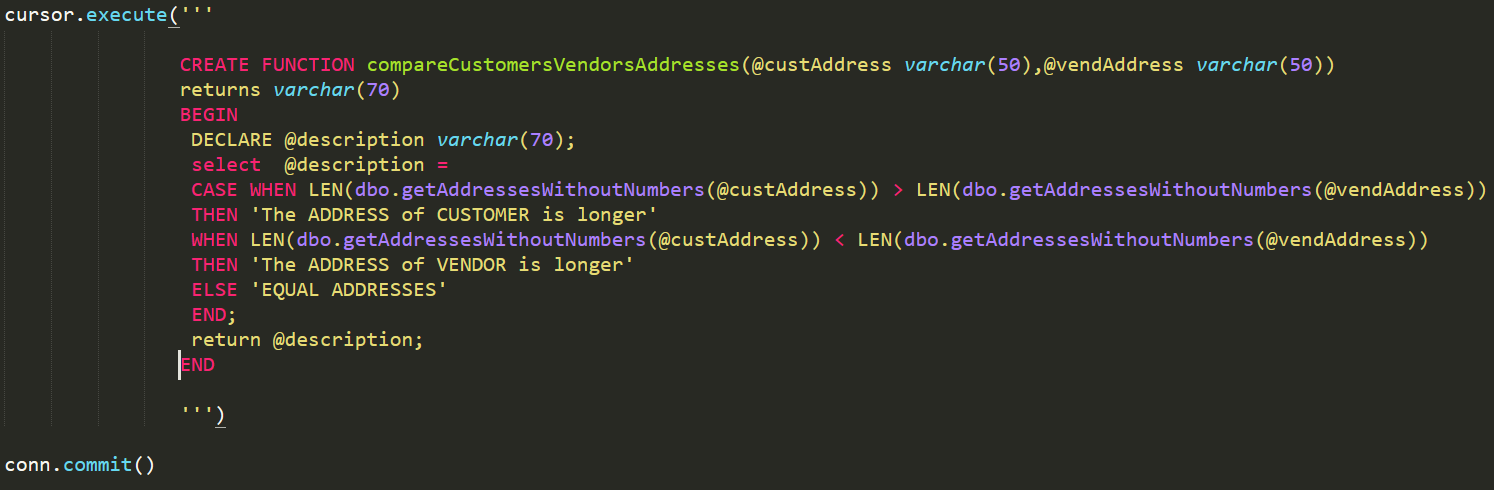
 **Be certain that we deleted the Function:**

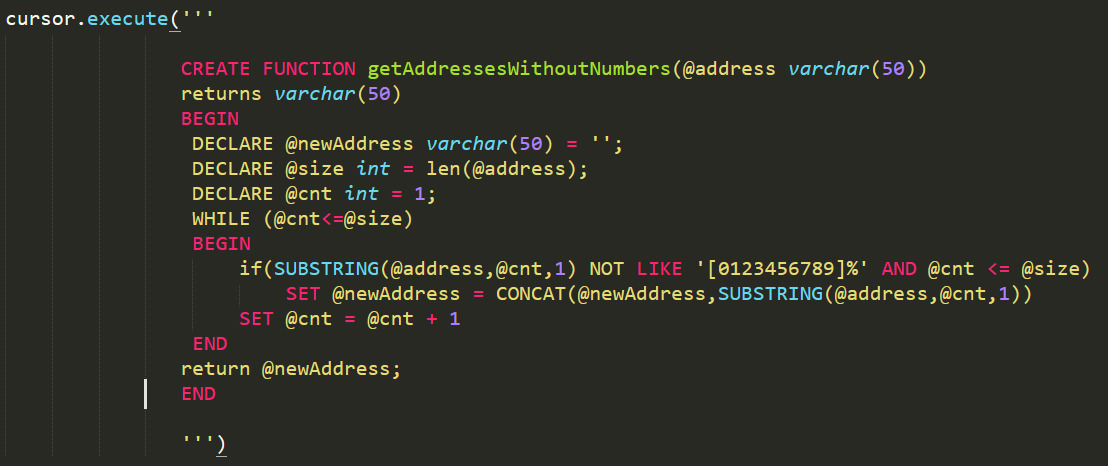
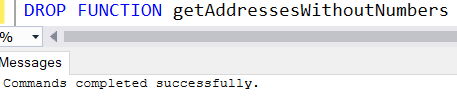
 **The Python code to execute the query:**

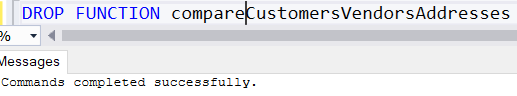
 **The output of the query in console:**

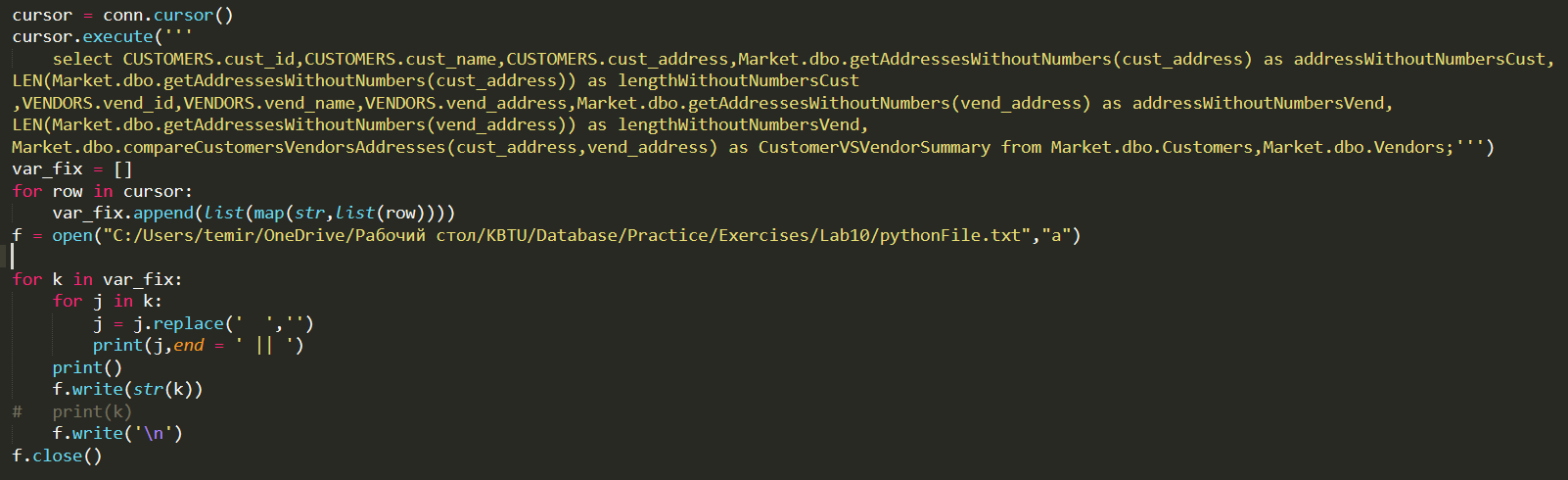
 **The output in txt file:**

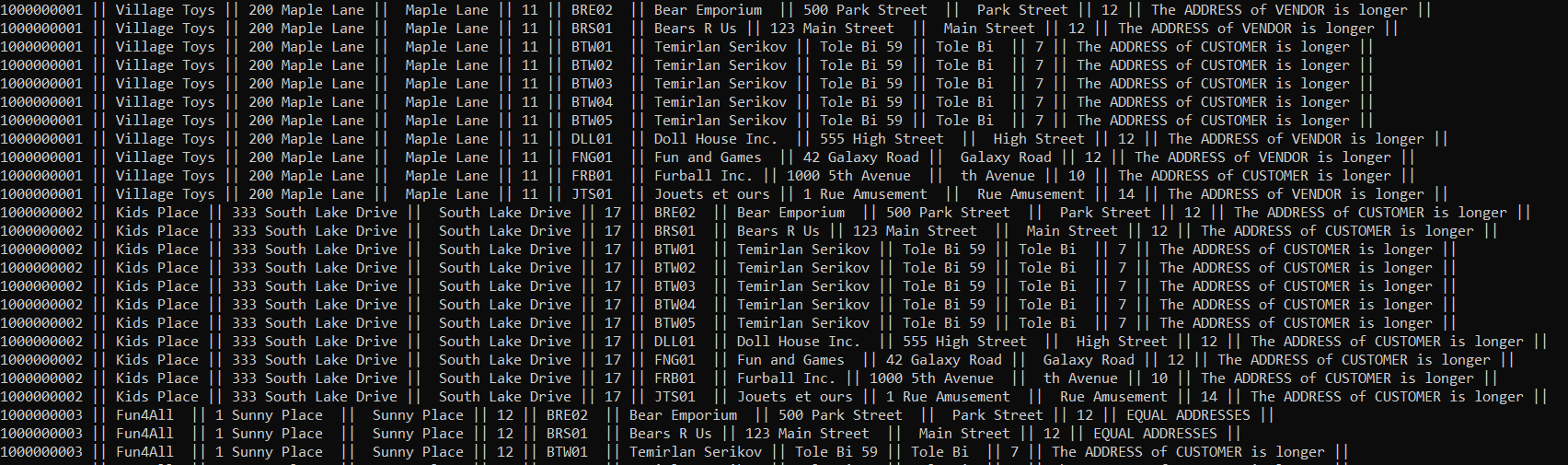
1. Write two functions where one will return the address for the supplied person without any figures while the second function would use it and compare the lengths of the addresses between customers and vendors and SHOWS The ADDRESS of CUSTOMER is longer if the length of the customer’s address without numbers is longer than vendor’s , The ADDRESS of VENDOR is longer if vendor’s if bigger or EQUAL ADDRESSES if they are the same. Compare all the customers to the vendors using CARTESIAN PRODUCT.

**The Python Functions creation:**

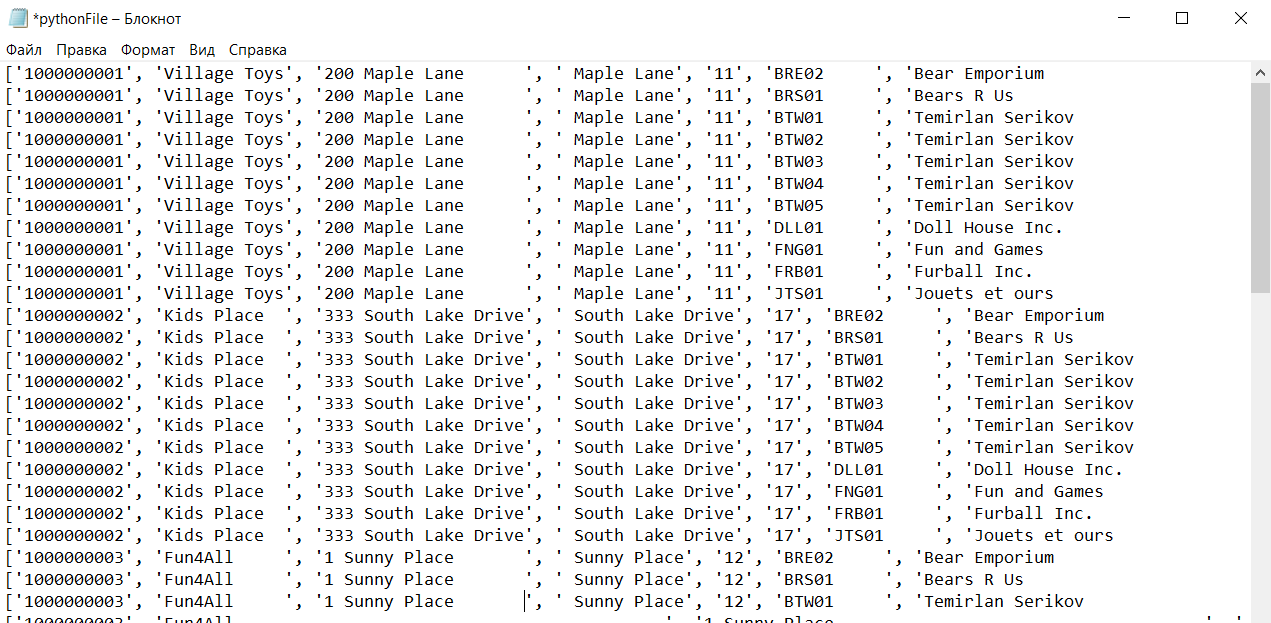
**Be sure that we deleted the function in advance:**

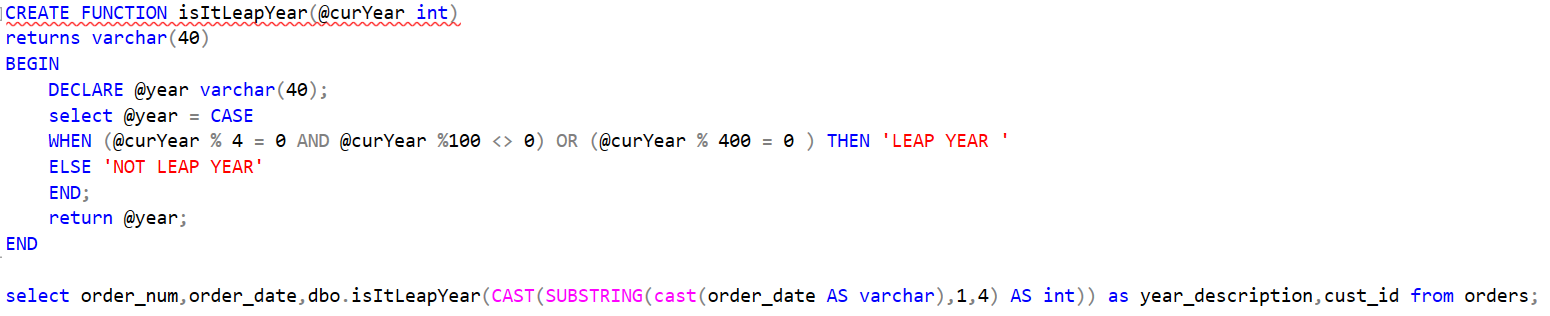
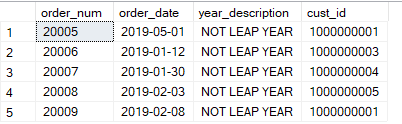


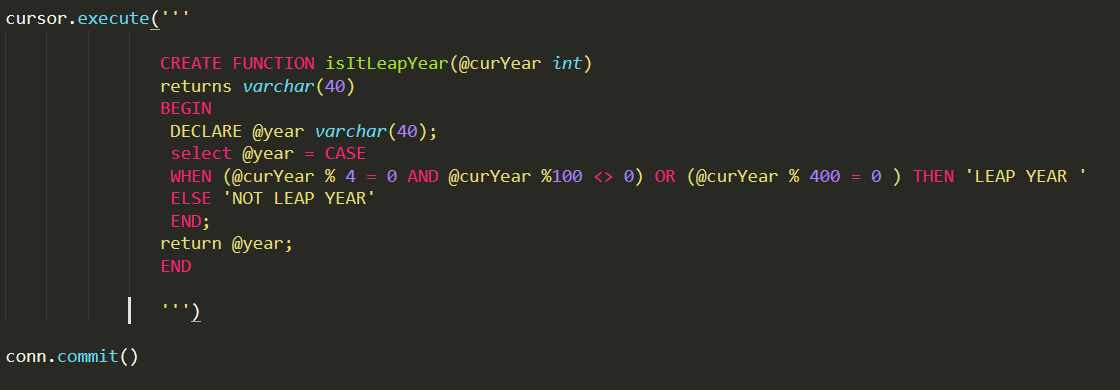
**The Python code to execute the queries with functions:**

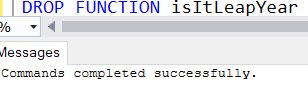
**The output in console:**

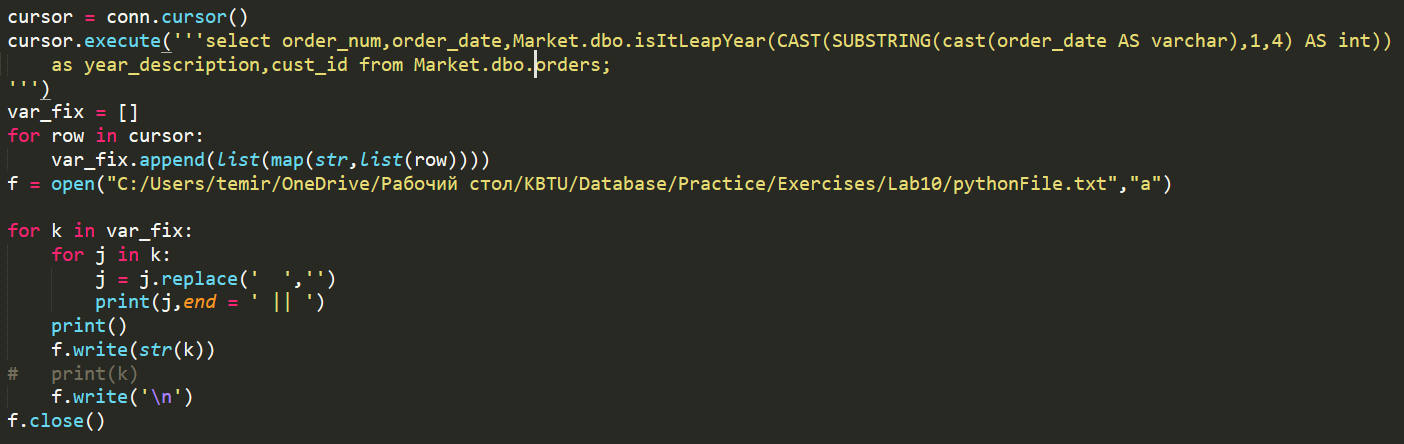
**The output in txt File:**



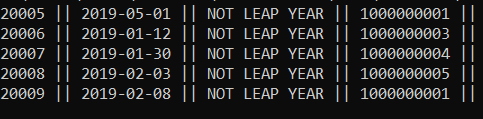
1. Write a function that would check the taken year and return ‘LEAP YEAR’ or ‘NOT LEAP YEAR’.

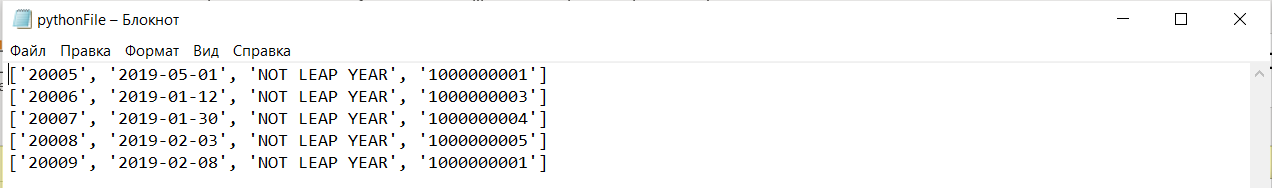
**The Python Creation Function code:**

**Be sure that we deleted the Function in advance:**

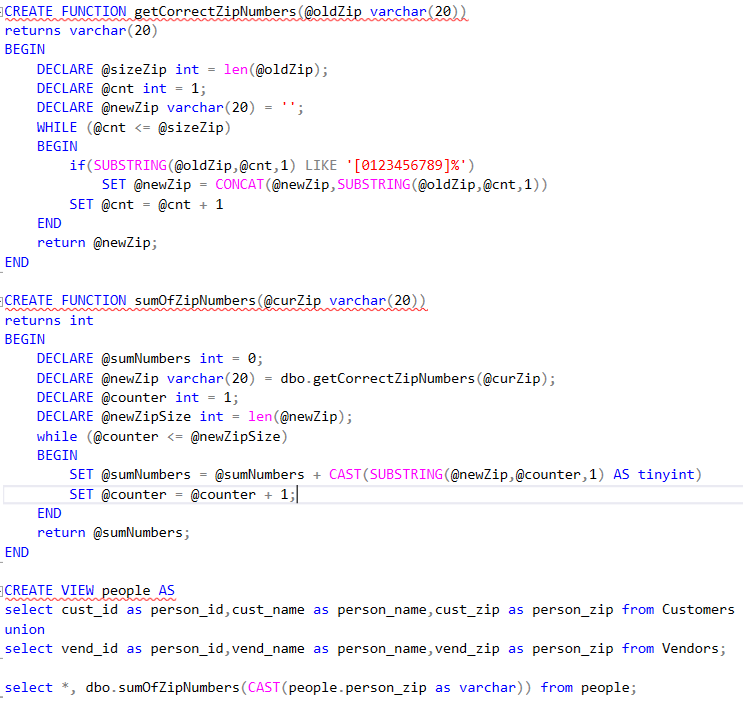
**Python code to execute the query with function calling:**

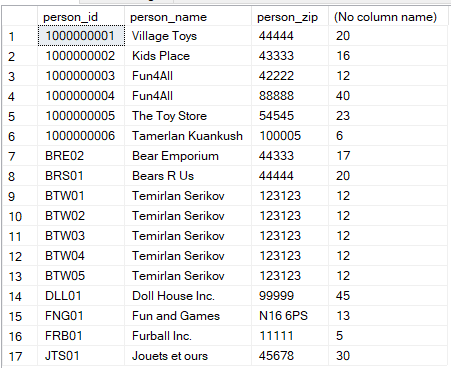
**The console output of the query:**



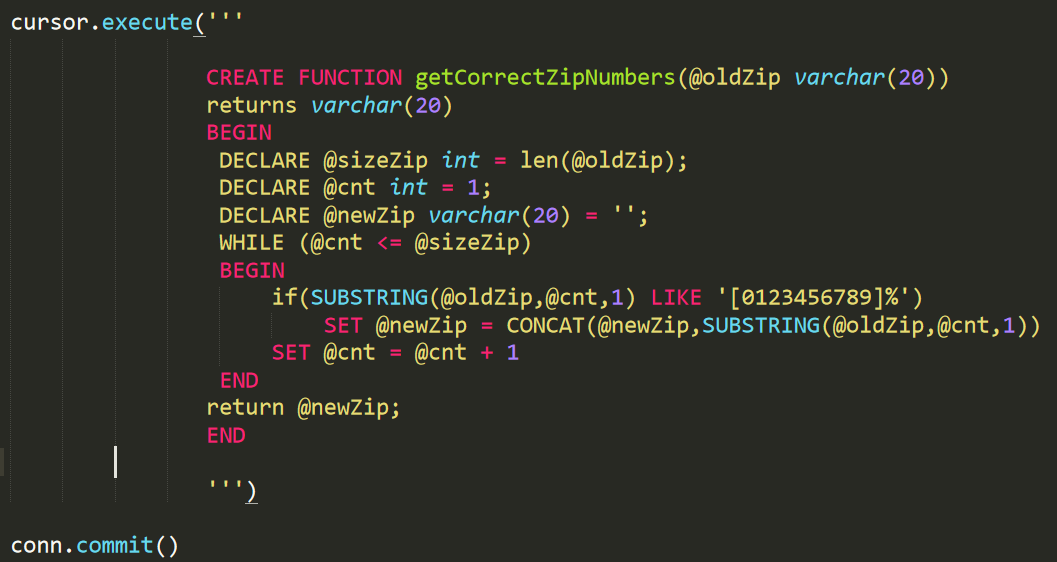
**The txt file output:**

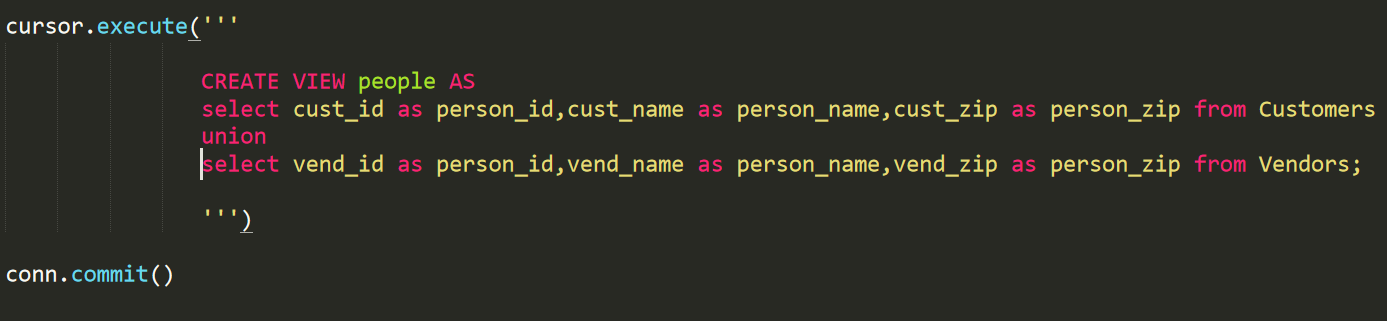
1. Write functions that together would produce the sum of the figures that are inside of the ZIP column but there problem can appear since there are some rows which have not only numbers but letters too. For example, the zip = N16 6PS would produce 13 (1 + 6 + 6). Produce the solution for all the Zip numbers among the whole people. You can create a view where you would store al the people (Customers + Vendors).



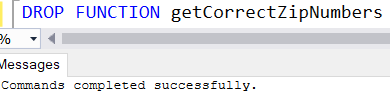
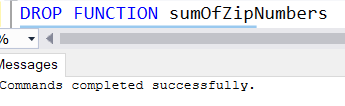


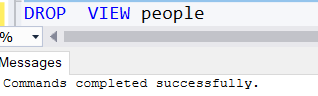
**The Python code to create Functions:**

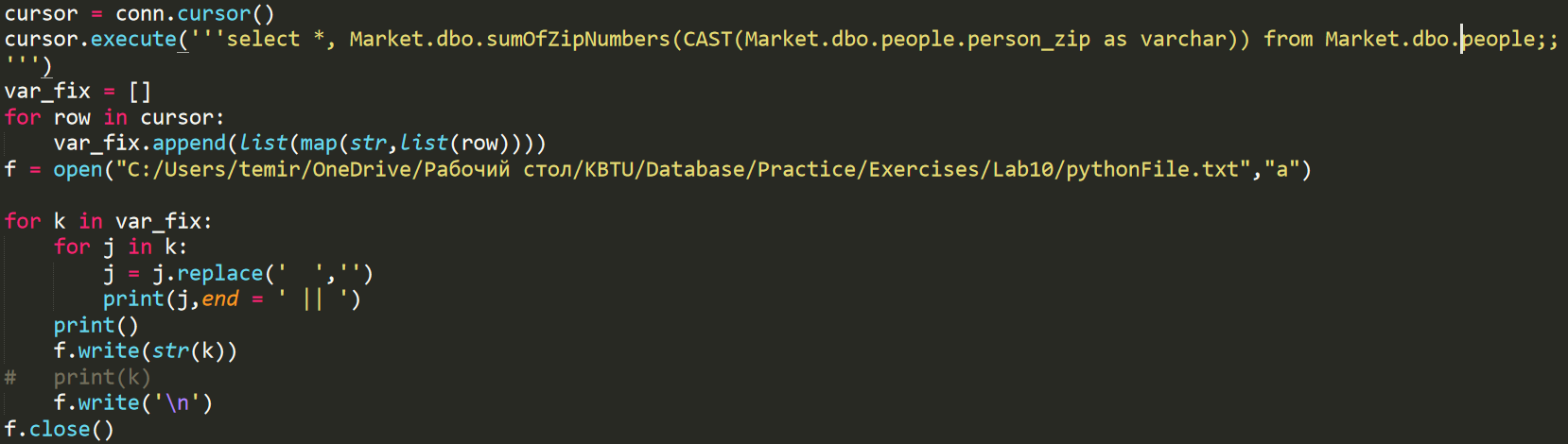




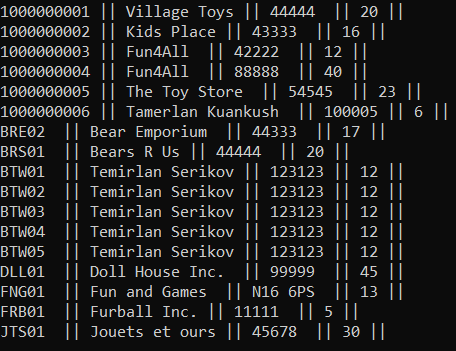
**Be sure that we deleted all the functions and view in advance:**

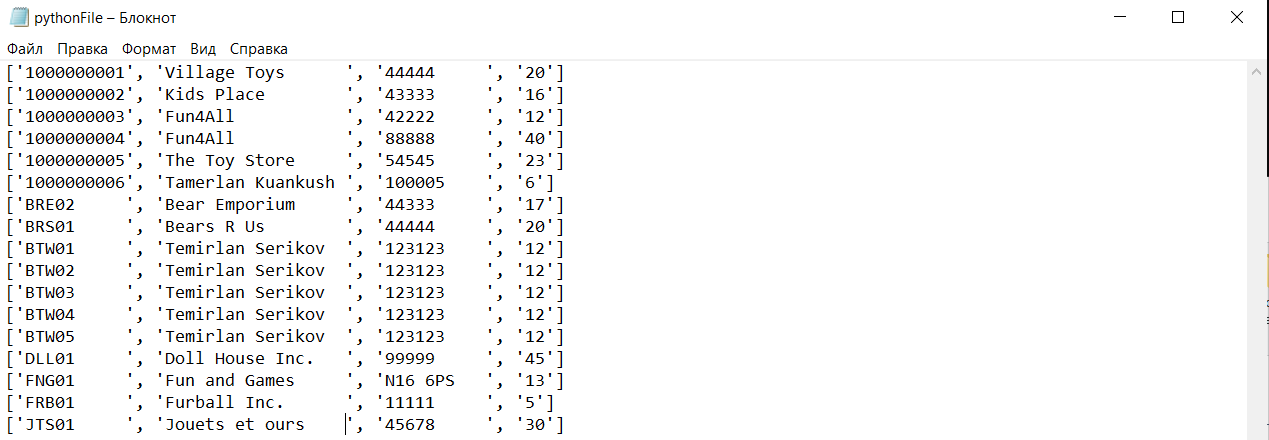




**The Python code to execute final query:**

**The Python output of the query in console:**



**The Output in txt File:**