**ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ PYTHON**

Βασίλης Μηλιώνης

ΑΜ: 1067415

Περιεχόμενα

[Μέρος Α: Αποτελέσματα Κώδικα 2](#_Toc73570782)

[Μέρος Β: Σχόλια – Παραδοχές 5](#_Toc73570783)

[Μέρος Γ: Κώδικας - Web Scraping 6](#_Toc73570784)

[Μέρος Δ: Κώδικας - Plot using Matplotlib 9](#_Toc73570785)

[Μέρος Ε: Κώδικας - Αποθήκευση σε βάση MySQL 11](#_Toc73570786)

[Μέρος ΣΤ: Η συνάρτηση main 13](#_Toc73570787)

### Μέρος Α: Αποτελέσματα Κώδικα

Αρχεία CSV:

Αφού ο κώδικας web\_scraping κατεβάσει τα αρχεία με τα δεδομένα, τα αποσυμπιέσει και να μεταφέρει σε αρχεία .txt, διαχειρίζομαι κατάλληλα τα .txt, μέσω της συνάρτησης tsv\_to\_csv() και τα αποθηκεύω σε ένα αρχείο .csv . Το παρακάτω στιγμιότυπο περιέχει ένα μέρος από τα δεδομένα του πρώτου αρχείου (κωδικός στο Eurostat: tin00174).

Εικόνα που περιέχει πίνακας

Περιγραφή που δημιουργήθηκε αυτόματα

Γραφήματα:

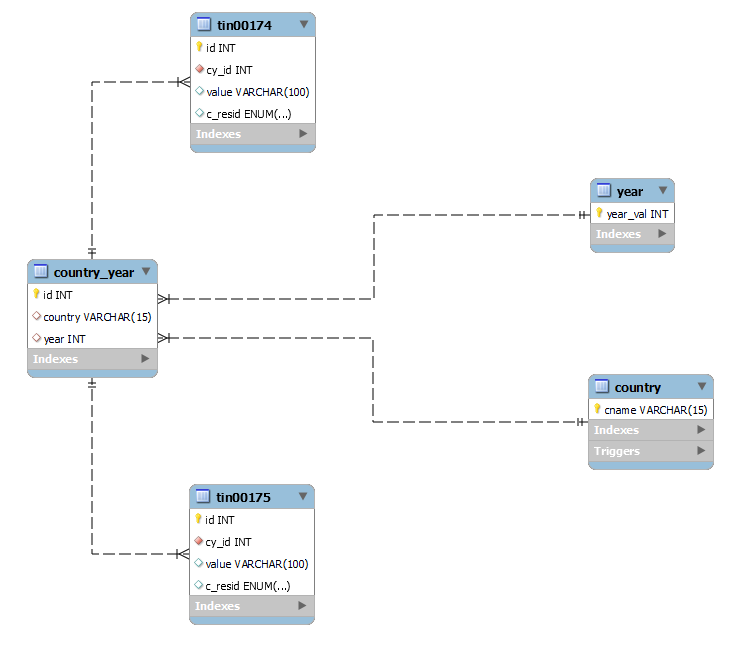
Αφού πλέον τα δεδομένα μου είναι αποθηκευμένα σε αρχεία .csv μπορώ να εκμεταλλευτώ τη μορφή τους και να φορτώσω κάποια από αυτά σε στιγμιότυπα της κλάσης CountryStats (η κλάση Stats οργανώνει κάπως καλύτερα τα δεδομένα, ώστε να μη διαχειρίζομαι λίστες 2 διαστάσεων, μπορούσε να παραληφθεί). Μέσω των της μεγάλης κλάσης (CountryStats) φτιάχνω συναρτήσεις που κάνουν plot τα δεδομένα που περιέχονται αποθηκευμένα μέσα στο κάθε στιγμιότυπο. Η plot\_me() ορίστηκε (επειδή δε διάβασα καλά την εκφώνηση) και φτιάχνει bar charts με όλα τα δεδομένα (“FOR”, “NAT”, “TOTAL”) για τη χώρα, ενώ η request\_plot() δημιουργεί τα 4 plot που ζητώνται.

|  |  |
| --- | --- |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

Σχηματικό Βάσης:

Πέρα από το σχηματικό που παραθέτω παρακάτω, στα παραδοτέα έχω ένα φάκελο με όνομα pythonDB. Μέσα σε αυτόν υπάρχουν αρχεία .sql που παράχθηκαν μέσω του MySQL Workbench (Server ⇒ Data Export ⇒ Επιλογή της βάσης ⇒ Start Export). Αυτά μπορούν να φορτωθούν μέσω του ίδιου προγράμματος, αφού δημιουργηθεί (και επιλεχθεί) η βάση με το όνομα pythonDB, με την διαδικασία (Server ⇒ Data Import ⇒ Επιλογή του φακέλου ⇒ Start Import).



### Μέρος Β: Σχόλια – Παραδοχές

Όπως αναφέρω και παρακάτω χρησιμοποιήθηκε η βιβλιοθήκη selenium αντί της urllib, προκειμένου να κάνω το web scraping. Παρόλα αυτά, δεν γνωρίζω αν είναι απόλυτα αυτό που ζητείται στην εκφώνηση και για αυτό το αναφέρω. Αλλάζοντας η τιμή της εντολής (chrome\_options.headless = True, αρχείο web\_scraping) σε False και τρέξετε το script, θα δείτε ακριβώς ότι το script αναζητεί στη Eurostat τον ακριβή τίτλο των αρχείων που μας πρέπει να επεξεργαστούμε και κατεβάζει τα 2 συμπιεσμένα αρχεία με τα δεδομένα. Συνεπώς δεν παίρνω τα δεδομένα κατευθείαν από τον πίνακα της ιστοσελίδας, κάτι που επέλεξα συνειδητά λόγω του ότι η javascript που υπάρχει στο site θα έκανε αυτή τη διαδικασία πολύ δύσκολη και χρονοβόρα.

Έπειτα σε κανένα σημείο του κώδικα δεν έχω φροντίσει για κακό user input και exception που μπορούν να εμφανιστούν. Καθώς και η main δεν δίνει πολλές επιλογές μέσω των argument του script (π.χ. αλλαγή των στοιχείων του χρήστη για σύνδεση στη βάση δεδομένων). Οι μόνες επιλογές που δίνονται είναι μέσω του πρώτου ορίσματος (download), αν θα ξανακατεβάσετε όλα τα αρχεία από τη Eurostat, οπότε αυτόματα θα διαγραφούν τα παλιά δεδομένα και θα αντικατασταθούν από τα νέα. Αν το 2ο argument είναι (sql) θα γίνει update και η βάση δεδομένων. Μέσω των υπόλοιπων ορισμάτων μπορείτε να δώσετε τον κωδικό χωρών που επιθυμείτε να πάρετε τα plot.

Παραδείγματα εκτέλεσης:

1. main.py download sql EL
2. main.py download EL
3. main.py sql EL
4. main.py EL CY UK DE

### Μέρος Γ: Κώδικας - Web Scraping

Καταρχάς αναφέρουμε ότι η διαδικασία του web-scraping έγινε μέσω της βιβλιοθήκης selenium, αφού η ιστοσελίδα της Eurostat λειτουργεί κυρίως μέσω javascript. Αυτό κάνει την ιστοσελίδα διαδραστική, με αποτέλεσμα να αλλάζει δυναμικά ο html και ο css κώδικας. Το selenium ήταν μια εύκολη εναλλακτική, οπότε τελικά δεν χρησιμοποίησα καθόλου την βιβλιοθήκη urllib.

1. # webdriver - used because eurostat is dynamic (that is, uses Javascript)
2. # so I need a way to parse the code before scraping - selenium is the easiest way I found
3. from selenium import webdriver
4. # interact with the website (enter, click, etc.)
5. from selenium.webdriver.common.keys import Keys
6. # extract the .tsv.gz file
7. import gzip
8. # interact with os
9. import shutil
10. import os
11. # sleep
12. import time
13. # regex
14. import re
15. # csv format library
16. import csv

19. # will unzip the file (that is in current directory) to a txt with the same filename
20. # will also delete the compressed file
21. def unzip\_gz(filename, extension):
22. cur\_dir = os.getcwd()
23. # unzip the file
24. tin\_file = cur\_dir + '/' + filename + extension
25. with gzip.open(tin\_file, 'r') as file\_in:
26. with open(filename + '.txt', 'wb') as file\_out:
27. shutil.copyfileobj(file\_in, file\_out)
28. # remove the zip you downloaded with selenium
29. os.remove(tin\_file)

32. # Initialize a driver with some settings
33. # and launches the desired url
34. def start\_driver():
35. # path to the driver selenium will use
36. path = "chromedriver.exe"
37. # Changing the download folder, it is only valid for current session
38. cur\_dir = os.getcwd()
39. chrome\_options = webdriver.ChromeOptions()
40. # headless -> do not open a chrome browser as a window
41. chrome\_options.headless = True
42. prefs = {'download.default\_directory': cur\_dir}
43. chrome\_options.add\_experimental\_option('prefs', prefs)
44. # initializing the driver
45. driver = webdriver.Chrome(path, options=chrome\_options)
46. url = "https://ec.europa.eu/eurostat/web/main/data/database"
47. driver.get(url)
48. return driver

51. # Uses the open driver and navigates in eurostat
52. # uses eurostat search-bar in order to locate the files with title tile
53. # downloads the files and goes back to where it started, so it multiple calls are possible in the same session
54. def eurostat\_search(driver, title):
55. # get to the search bar in eurostat (the name of which is text in the html code)
56. search = driver.find\_element\_by\_name("text")
57. # types in the search bar the file I am looking for
58. search.send\_keys(title)
59. # press enter (to search)
60. search.send\_keys(Keys.RETURN)
61. # finds the box with the correct title
62. downloader = driver.find\_element\_by\_link\_text(title)
63. # clicks that title
64. downloader.click()
65. # finds the link that will give me the file
66. downloader = driver.find\_element\_by\_link\_text("Download table")
67. # click on the element (download button, which will download a .tsv.gz file
68. downloader.click()
69. # get the code name of the dataset you are downloading
70. product\_section\_left\_list = driver.find\_elements\_by\_class\_name("product-section-left")
71. code = ""
72. for product\_section\_left in product\_section\_left\_list:
73. temp = product\_section\_left.text
74. if temp.find("Code: ") != -1:
75. code = temp.split("\n")[0].split(" ")[1]
76. break
77. # If code is empty then, I don't have the name of the file I downloaded - thus I need to throw
78. # TODO: Get the filename (code), just from the contents of the folder (matching the extension)
79. if code == "":
80. print("Did not manage to get the code of the dataset!")
81. exit(1)
82. driver.back()
83. driver.back()
84. return code

87. # closes the driver, given in arguments, but before doing so,
88. # it ensures that the files in list filename downloaded successfully
89. def close\_driver(driver, filenames, extension):
90. flag = 0
91. # Wait until expected file is downloaded and in current path (max 10 sec)
92. for filename in filenames:
93. time\_cnt = 0
94. while os.path.exists(os.getcwd() + '/' + filename + extension) != 1:
95. time.sleep(2)
96. time\_cnt += 2
97. if time\_cnt >= 10:
98. # close the tab you opened
99. driver.close()
100. print(f"File: {filename+extension} did not download!")
101. flag = 1
102. exit(1)
103. driver.close()

106. # web\_scraping main -> Will download and decompress files
107. # The result is txt files in directory Web\_Scraping\_Results
108. def get\_files(ws\_argv):
109. # If directory already exists delete it
110. if os.path.exists("Web\_Scraping\_Results") == 1:
111. # Note: rmtree fails if folder tree contains read-only files
112. shutil.rmtree("Web\_Scraping\_Results")
113. os.mkdir("Web\_Scraping\_Results")
114. # Create a list in which I will store the names of the files I downloaded
115. filenames = []
116. extension = ".tsv.gz"
117. driver = start\_driver()
118. # iterate through the requested files (IMPORTANT: the titles should be exact!)
119. # search for them and download 'em
120. for arg in ws\_argv:
121. filename = eurostat\_search(driver, arg)
122. filenames.append(filename)
123. close\_driver(driver, filenames, extension)
124. # unzip files
125. for filename in filenames:
126. unzip\_gz(filename, ".tsv.gz")
127. # Move files to directory Web\_Scraping\_Results, so everything is organized
128. os.rename(filename + ".txt", "Web\_Scraping\_Results/" + filename + ".txt")
129. full\_path = "Web\_Scraping\_Results/" + filename + ".txt"
130. # Do some data manipulation and save data in a csv fromat
131. tsv\_to\_csv(full\_path)
132. # Delete the .txt files (since you now have the .csv)
133. os.remove(full\_path)
134. # return the list of names so you may later manipulate the data
135. return filenames

138. # ------------------------------------------------ Data manipulation ------------------------------------------------- #
140. # Gets the path of a file and splits it into a list
141. # 0-> path | 1-> filename | 3-> extension
142. def spilt\_filename(full\_path):
143. # get the path of the file (if included)
144. path = re.findall(r"(.\*/+)\*", full\_path)[0]
145. # remove that path from the file name
146. file\_with\_ext = re.sub(r"(.\*/+)\*", "", full\_path)
147. # get the extension of the file (assuming there is only one)
148. ext = re.findall(r"(\..\*)", file\_with\_ext)[0]
149. # ignore the extension
150. f\_name = re.findall(r".\*(?=\.)", file\_with\_ext)[0]
151. path\_list = [path, f\_name, ext]
152. return path\_list

155. # manipulate the first line
156. def csv\_fix\_first\_line(first\_line):
157. # the first columns in the tsv file are being seperated by commas
158. first\_line = first\_line.split(',')
159. # a stupid way to do resident = "c\_resid"
160. resident = first\_line[0]
161. # a stupid way to do geo = "geo"
162. geo = re.findall(r"(\w+)(?=\\)", first\_line[3])[0]
163. # the rest of the columns are seperated by tabs
164. years = first\_line[3].split("\t")
165. # the fist argument is "geo/time", not a year so I remove it
166. years.pop(0)
167. # convert everything else into integers
168. years = [int(year) for year in years]
169. # target first line: "c\_resid", "geo", 2008, 2009, etc.
170. years.insert(0, resident)
171. years.insert(1, geo)
172. return years

175. # Convert the tsv file you downloaded into a csv file
176. def tsv\_to\_csv(file):
177. # open input file (tsv)
178. inp = open(file, 'r')
179. # open output file (csv)
180. path\_list = spilt\_filename(file)
181. out = open(path\_list[0]+path\_list[1]+".csv", 'w')
182. csv\_out = csv.writer(out, delimiter=',', quoting=csv.QUOTE\_NONNUMERIC)
183. # first line of csv (different from the rest - not really)
184. first\_line = inp.readline()
185. first\_line = csv\_fix\_first\_line(first\_line)
186. csv\_out.writerow(first\_line)
187. # the rest of the csv file
188. lines = inp.readlines()
189. for line in lines:
190. labels = line.split(',')
191. residency = labels[0]
192. country = labels[3].split('\t').pop(0)
193. # values is a list that will contain all arithmetic values of a specific row
194. values = re.findall(r"(\d+|:)[\s]", re.sub(country, "", labels[3]))
195. if len(values) != 12:
196. print(f"Data corrupted! First wrong line:\n{line}")
197. exit(1)
198. # Ternary Operator combined with one line for loop - python is weird
199. values = [value if value == ':' else int(value) for value in values]
200. # Append info at the beggining of each csv line
201. values.insert(0, residency)
202. values.insert(1, country)
203. csv\_out.writerow(values)
204. # close mi files
205. inp.close()
206. out.close()

### Μέρος Δ: Κώδικας - Plot using Matplotlib

1. import web\_scraping as ws
2. import csv
3. import matplotlib.pyplot as plt
4. import numpy as np
5. import os
6. from textwrap import wrap
8. # matplotlib style
9. plt.style.use('fivethirtyeight')

12. # Load stats from file file of resident c\_resid ("FOR", "NAT", "TOTAL"), country geo, for years in range of
13. # start\_year and end\_year
14. def load\_stats(file, c\_resid, geo, start\_year, end\_year):
15. path\_list = ws.spilt\_filename(file)
16. inp = open(path\_list[0] + path\_list[1] + ".csv", 'r')
17. csv\_inp = csv.DictReader(inp)
18. for row in csv\_inp:
19. if row["c\_resid"] == c\_resid and row["geo"] == geo:
20. # Do not forget that range counts from 0 (thus +1 is required if you need to get the last year)
21. years = list(range(int(start\_year), int(end\_year + 1), 1))
22. values = []
23. for year in years:
24. values.append(row[str(year)])
25. res = Stats(years, values)
26. return res

29. # Will save every stat for a specific topic of a certain country
30. class CountryStats:
31. def \_\_init\_\_(self, country, file, start\_year, end\_year, description="You forgot description"):
32. self.country = country
33. self.description = description
34. self.foreign = load\_stats(file, "FOR", country, start\_year, end\_year)
35. self.native = load\_stats(file, "NAT", country, start\_year, end\_year)
36. self.total = load\_stats(file, "TOTAL", country, start\_year, end\_year)
38. # Plots all the info the object contains into a bar graph
39. def plot\_me(self):
40. # width of the whole group of bars
41. width = 0.8
42. # number of groups (of bars)
43. num\_bar = len(self.native.years)
44. # create a linear space so I may plot a group on each point
45. x = np.arange(num\_bar)
46. # customize the figure
47. all\_plot, ap = plt.subplots()
48. all\_plot.set\_figheight(7)
49. all\_plot.set\_figwidth(15)
50. # plot the subplots (group of bars)
51. ap.bar(x - width/3, self.native.get\_int\_values(), width=width/3, label="Natives")
52. ap.bar(x, self.foreign.get\_int\_values(), width=width/3, label="Foreigners")
53. ap.bar(x + width/3, self.total.get\_int\_values(), width=width/3, label="Total")
54. # labels - legend - title
55. plt.xlabel("Year")
56. plt.ylabel(f"Number of {self.description.split(' ')[0]} ")
57. ap.legend()
58. # HOW TO WRAP A BIG TITLE: https://stackoverflow.com/questions/10351565/how-do-i-fit-long-title
59. plt.title("\n".join(wrap(f"{self.description} in {self.country}")))
60. # set x ticks to be the linear space I created earlier
61. ap.set\_xticks(x)
62. # change the values that are visible on the x axis of the plot
63. ap.set\_xticklabels(self.total.years)
64. fig\_path = "final\_plots/"
65. fig\_name = self.country + '\_' + self.description.split(" ")[0] + ".jpg"
66. if os.path.exists(fig\_path) == 0:
67. os.mkdir(fig\_path)
68. plt.savefig(fig\_path + fig\_name)
69. Plt.close()
71. # Function that saves the plots professor requested, using
72. # options: "Natives", "Foreigners" or "Total"
73. def request\_plot(self, option):
74. plt.figure(figsize=[14, 12])
75. if option == "Natives":
76. safe\_obj = make\_safe(self.native)
77. plt.plot(safe\_obj.years, safe\_obj.get\_int\_values(), label="Natives")
78. elif option == "Foreigners":
79. safe\_obj = make\_safe(self.foreign)
80. plt.plot(safe\_obj.years, safe\_obj.get\_int\_values(), label="Foreigners")
81. elif option == "Total":
82. safe\_obj = make\_safe(self.total)
83. plt.plot(safe\_obj.years, safe\_obj.get\_int\_values(), label="Total")
84. else:
85. print("Not a valid input (func: plot\_as\_req)")
86. exit(1)
87. # fix x axis ticks
88. plt.xticks(self.native.years, self.native.years)
89. # labels - legend - title
90. plt.xlabel("Year")
91. plt.ylabel(f"Number of {self.description.split(' ')[0]} ")
92. plt.legend()
93. plt.title("\n".join(wrap(f"{self.description} in {self.country}")))
94. # Save fig
95. fig\_path = "final\_plots/"
96. fig\_name = self.country + '\_' + self.description.split(" ")[0] + '\_' + option + ".jpg"
97. plt.savefig(fig\_path + fig\_name)
98. plt.close()

101. # This function will clean the entire year from both the years list
102. # and the values list, of zero values, so the graphs are more accurate and don't have weird dips to 0
103. # The bar diagrams need to be of the same dimension, but on a certain year a country may have
104. # stats about foreigners and have none for natives. This means that the years I will try to plot on the same
105. # bar chart will differ. In order to make my life easier I will be using the function below, on demand, when
106. # I am about to plot the data into line charts
107. def make\_safe(stats\_obj):
108. cnt = 0
109. values = []
110. years = []
111. for value in stats\_obj.values:
112. if int(value) != 0:
113. values.append(stats\_obj.values[cnt])
114. years.append(stats\_obj.years[cnt])
115. cnt += 1
116. safe\_obj = Stats(years, values)
117. return safe\_obj

120. # Save the years and the values of the given stats
121. # Will be used 3 times (foreign, native, total) in the CountryStats class
122. class Stats:
123. def \_\_init\_\_(self, years, values):
124. self.years = years
125. self.values = self.make\_safe(values)
127. def get\_int\_values(self):
128. res = [int(value) for value in self.values]
129. return res
131. def print\_stats(self):
132. print(f"Years: {self.years}")
133. print(f"Values: {self.values}")
135. # overrides the function above (an egrafes c++).
136. # Some of the values in the csv file are ":", which means that there are no available data
137. # In order to be able to plot countries that may have a column like that
138. # I will make\_safe the data, by replacing ":" with the integer 0
139. def make\_safe(self, values):
140. cnt = 0
141. ret = []
142. for value in values:
143. if value != ':':
144. ret.append(value)
145. else:
146. ret.append(0)
147. cnt += 1
148. return ret

### Μέρος Ε: Κώδικας - Αποθήκευση σε βάση MySQL

1. import mysql.connector
2. import csv
3. import re
4. import web\_scraping as ws

7. def connect\_mysql():
8. # establish connection to the local db
9. # the user has root privilages
10. my\_db = mysql.connector.connect(
11. host="localhost",
12. user="ceid",
13. passwd="1SafePass"
14. )
15. mycursor = my\_db.cursor()
16. # drop the old database
17. mycursor.execute("DROP DATABASE IF EXISTS pythonDB;")
18. # create a db to connect to
19. mycursor.execute("CREATE DATABASE IF NOT EXISTS pythonDB;")
20. # use the db
21. mycursor.execute("USE pythonDB")
22. return my\_db

25. # create all those tables that are constant, no matter which file you load
26. # the drops are not required, since I drop the whole base on connection
27. def create\_default\_tables(connection):
28. mycursor = connection.cursor()
29. mycursor.execute("DROP TRIGGER IF EXISTS country\_year\_after\_country")
30. mycursor.execute("DROP TABLE IF EXISTS country\_year;")
31. mycursor.execute("DROP TABLE IF EXISTS year;")
32. mycursor.execute("DROP TABLE IF EXISTS country;")
34. mycursor.execute("CREATE TABLE country ("
35. "cname VARCHAR(15) NOT NULL,"
36. "PRIMARY KEY (cname)"
37. ");")
38. mycursor.execute("CREATE TABLE year ("
39. "year\_val INT NOT NULL,"
40. "PRIMARY KEY (year\_val)"
41. ");")
42. mycursor.execute("CREATE TABLE country\_year ("
43. "id INT AUTO\_INCREMENT PRIMARY KEY,"
44. "country VARCHAR(15),"
45. "FOREIGN KEY (country) REFERENCES country(cname)"
46. "ON UPDATE CASCADE,"
47. "year INT,"
48. "FOREIGN KEY (year) REFERENCES year(year\_val)"
49. "ON UPDATE CASCADE"
50. ");")
51. # trigger that will insert all records into the country\_year table
52. mycursor.execute("CREATE TRIGGER country\_year\_after\_country "
53. "AFTER INSERT ON country "
54. "FOR EACH ROW "
55. " BEGIN "
56. " DECLARE done INT DEFAULT FALSE;"
57. " DECLARE y INT;"
58. " DECLARE cur CURSOR FOR SELECT \* FROM year ;"
59. " DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;"
60. " OPEN cur;"
61. " year\_loop: LOOP "
62. " FETCH cur INTO y;"
63. " IF done THEN "
64. " LEAVE year\_loop;"
65. " END IF;"
66. " INSERT INTO country\_year (country, year) VALUES (NEW.cname, y);"
67. " END LOOP;"
68. " CLOSE cur;"
69. "END")

72. # the only dynamic thing about them is the name
73. # I chose varchar(100) to save the data so I don't have to remove ":"
74. def create\_dynamic\_table(connection, table\_name):
75. mycursor = connection.cursor()
76. mycursor.execute(f"DROP TABLE IF EXISTS {table\_name}")
77. mycursor.execute(f"CREATE TABLE {table\_name} ("
78. f"id INT AUTO\_INCREMENT PRIMARY KEY,"
79. f"cy\_id INT NOT NULL,"
80. f"value VARCHAR(100),"
81. f"c\_resid ENUM('FOR', 'NAT', 'TOTAL'),"
82. f"FOREIGN KEY (cy\_id) REFERENCES country\_year(id)"
83. f"ON UPDATE CASCADE"
84. f");")

87. # given a connection, a table and the record I want to insert returns
88. # true is record not in the table and false if record already exists in the table
89. # TODO: FIX: new\_record must be a string, fix it so it may be a list (not important)
90. def can\_insert(connection, table\_name, new\_record):
91. mycursor = connection.cursor()
92. mycursor.execute(f"SELECT \* FROM {table\_name}")
93. records = mycursor.fetchall()
94. for record in records:
95. for value in record:
96. if str(value) == str(new\_record):
97. return 0
98. return 1

101. # returns the id which corresponds to country, year record in coutry\_year table
102. # could be avoided if I only inserted on the dynamic tables (tin00174, tin00175)
103. # with a JOIN
104. def get\_country\_year\_id(connection, country, year):
105. mycursor = connection.cursor()
106. mycursor.execute("SELECT id FROM country\_year WHERE country = %s AND year = %s", (country, year))
107. return mycursor.fetchone()[0]

110. # remember that the values need to be in a tuple form
111. # the function below loads all tables using the data in filename(csv format)
112. def load\_tables(connection, filename):
113. # use the filename (ex. tin00174) in order to name the table you are creating
114. fname = ws.spilt\_filename(filename)[1]
115. create\_dynamic\_table(connection, fname)
116. # get the cursor that you will execute sql commands with
117. mycursor = connection.cursor()
118. inp = open(filename, 'r')
119. csv\_inp = csv.DictReader(inp)
120. # insert years into sql table year
121. # clean the first line (remove the 2 leftmost strings "c\_resid", "geo" and the '\n' at the end of the line)
122. years = inp.readline().split(',')[2:]
123. years = [re.sub(r"\n", "", year) for year in years]
124. # insert the years into the correct table (if not already inserted)
125. for year in years:
126. if can\_insert(connection, "year", year):
127. mycursor.execute("INSERT INTO year (year\_val) VALUES (%s)", (year,))
128. # get back to the beggining of the file
129. inp.seek(0)
130. # for each line (except the first one, csv library)
131. for row in csv\_inp:
132. cname = row["geo"]
133. # insert the country names into the correct table (if not already inserted)
134. if can\_insert(connection, "country", cname):
135. mycursor.execute("INSERT INTO country (cname) VALUES (%s)", (cname,))
136. # insert the data into the dynamic table I created earlier in this function
137. for year in years:
138. cy\_id = get\_country\_year\_id(connection, cname, year)
139. mycursor.execute(f"INSERT INTO {fname} (cy\_id, value, c\_resid) VALUES (%s, %s, %s)", (int(cy\_id), str(row[str(year)]), str(row["c\_resid"])))
140. # save any changes I made (only need to commit once on each session)
141. connection.commit()
142. # close your files!
143. inp.close()

### Μέρος ΣΤ: Η συνάρτηση main

1. import sys
2. import pickle
3. import web\_scraping as ws
4. import load\_data as ld
5. import proj\_sql as ps

8. def main():
9. # The titles of the datasets I want to download
10. ws\_argv = "Arrivals of residents/non-residents at tourist accommodation establishments",\
11. "Nights spent at tourist accommodation establishments by residents/non-residents"
12. dflag = 0
13. # Download if requested (first argument must be download)
14. if len(sys.argv) > 1 and sys.argv[1] == "download":
15. dflag = 1
16. filenames = ws.get\_files(ws\_argv)
17. # save the filenames in a pickle file so I may retrieve later
18. file = open("my\_pickle.txt", 'wb')
19. pickle.dump(filenames, file)
20. file.close()
22. # Update the database with the new data, if requested (second, or first argument sql)
23. if (len(sys.argv) > 2 and dflag == 1 and sys.argv[2] == "sql") or sys.argv[1] == "sql":
24. dflag += 1
25. mycursor = ps.connect\_mysql()
26. ps.create\_default\_tables(mycursor)
27. ps.load\_tables(mycursor, "Web\_Scraping\_Results/tin00174.csv")
28. ps.load\_tables(mycursor, "Web\_Scraping\_Results/tin00175.csv")
30. # load the filenames from pickle file
31. file = open("my\_pickle.txt", 'rb')
32. filenames = pickle.load(file)
33. file.close()
34. path = "Web\_Scraping\_Results/"
36. # DEBUG CODE:
37. # # If less than 3 arguments get to default (ELLADA, CYPRUS)
38. # # (There should always be at least 3 arguments arg0->main.py arg1-> requested country arg2-> req. country)
39. # # I only did this so I make compile easier with Shift + F10
40. # if len(sys.argv) < 3 and flag == 0:
41. # print("Defaulting to EL, CY...")
42. # args = ["EL", "CY"]
43. # else:
44. # args = sys.argv[flag+1:]
45. args = sys.argv[dflag+1:]
47. # Loop for each country
48. for geo in args:
49. cnt = 0
50. # Loop for each dataset title
51. for argv in ws\_argv:
52. country = ld.CountryStats(geo, path+filenames[cnt]+".csv", 2008, 2019, argv)
53. # DEBUG\_CODE:
54. # print(f"\t{argv} in {country.country}")
55. # print("Foreign Stats:")
56. # country.foreign.print\_stats()
57. # print("Native Stats:")
58. # country.native.print\_stats()
59. # print("Total Stats:")
60. # country.total.print\_stats()
61. country.plot\_me()
62. country.request\_plot("Total")
63. country.request\_plot("Foreigners")
64. cnt += 1

67. if \_\_name\_\_ == "\_\_main\_\_":
68. main()