Name – **Onkar Kajrolkar**

CampbuzzId – **2020Sep@03308**

Email – **onkarkajrolkar7798@gmail.com**

**Web Based Case Study**

**LBJ Cycle-2**

1. **Synopsis –**

Web Development has been an increasing technology with engineers mastering the skills of frontend as well as backend. This Case study gave us the opportunity to experience the tip of the iceberg that is Web Development. The Frontend for this case study was built with HTML5 for web pages and CSS for styling of the said web pages. The server side code was written using Python by implementing the Flask module for setting up a local server. The HTML5 web pages are stored in the “templates” folder and the CSS styling file is stored in the “static” folder. The Flask will search for the templates and static folders for rendering the webpages and styling files respectively. Using python and the data received through the web pages, we can write to/read from a CSV file.

1. **Source Code –**
2. **“~/students-server.py” –**
3. from flask import \*
4. import csv
5. keys = ['Student ID', 'Student Name', 'Gender', 'Date of Birth', 'City', 'State', 'Email ID', 'Qualification', 'Stream']
6. app = Flask(\_\_name\_\_)
7. app.secret\_key = "wubbalubbadubdub"
8. @app.route("/")
9. def index():
10. return render\_template("index.html")
11. @app.route("/add", methods=["POST", "GET"])
12. def add():
13. if request.method == "POST":
14. result = list(request.form.values());
15. split = result[3].split('-')
16. result[3] = split[2]+"-"+split[1]+"-"+split[0]
17. with open('students.csv', 'a', newline='\n') as csv\_file:
18. csv\_writer = csv.writer(csv\_file)
19. csv\_writer.writerow(result)
20. flash("Student added to the Database!")
21. return render\_template("add-student.html")
22. @app.route("/search", methods=["POST", "GET"])
23. def search():
24. if request.method == "POST":
25. id = list(request.form.values())
26. print(id[0])
27. success = False
28. with open('students.csv', 'r') as csv\_file:
29. csv\_reader = csv.reader(csv\_file)
30. for line in csv\_reader:
31. if line[0] == id[0]:
32. details = line
33. success = True
34. if not success:
35. if id[0] == '':
36. flash("Please Enter ID", "info")
37. else:
38. flash(f"Student ID: {id[0]} not found, Please try again!", "info")
39. else:
40. result = dict(zip(keys, details))
41. return render\_template("search-student.html", data=True, result=result)
42. return render\_template("search-student.html", data=False, result="")
43. @app.route("/display")
44. def display():
45. with open('students.csv', 'r') as csv\_file:
46. csv\_reader = csv.reader(csv\_file)
47. return render\_template("display-student.html", result=csv\_reader, keys=keys)
48. if \_\_name\_\_=="\_\_main\_\_":
49. app.run()
50. **“~/templates/index.html” –**
51. <!DOCTYPE html>
52. <html>
53. <link rel="stylesheet" type="text/css" href="{{ url\_for('static', filename='style.css')}}">
54. <head>
55. <title>Student Dashboard</title>
56. </head>
57. <body>
58. <ul id="list" type="none" align="center">
59. <li><button class="button" onclick="window.location='add'">Add Student</button></li>
60. <li><button class="button" onclick="window.location='search'">Search Student</button></li>
61. <li><button class="button" onclick="window.location='display'">Display all Students</button></li>
62. </ul>
63. </body>
64. </html>
65. **“~/templates/add-student.html”**
66. <!DOCTYPE html>
67. <html>
68. <link rel="stylesheet" type="text/css" href="{{ url\_for('static', filename='style.css')}}">
69. <head>
70. <title>Add A Student</title>
71. </head>
72. <body>
73. <div id="entry" align="left">
74. <ul type="none">
75. <form action="" method="post">
76. <li>Student Id</li>
77. <li><input type="text" name="id" required></li>
78. <li>Student Name</li>
79. <li><input type="text" name="name" required></li>
80. <li>Gender</li>
81. <li><input type="radio" name="gender" value="Male" required>Male</li>
82. <li><input type="radio" name="gender" value="Female" required>Female</li>
83. <li>Date of Birth</li>
84. <li><input type="date" name="date" required></li>
85. <li>City</li>
86. <li><input type="text" name="city" required></li>
87. <li>State</li>
88. <li><input type="text" name="state" required></li>
89. <li>Email Id</li>
90. <li><input type="text" name="email" required></li>
91. <li>Qualification</li>
92. <li><input type="text" name="qual" required></li>
93. <li>Stream</li>
94. <li><input type="text" name="stream" required></li>
95. <li><input type="submit" class="submit" value="Submit"></li>
96. </form>
97. {% with messages = get\_flashed\_messages() %}
98. {% if messages %}
99. {% for message in messages %}
100. <li>{{messages}}</li>
101. {% endfor %}
102. {% endif %}
103. {% endwith %}
104. <li><button class="submit" onclick="window.location='/'">Home</button></li>
105. </ul>
106. </div>
107. </body>
108. </html>
109. **“~/templates/search-student.html” –**
110. <head>
111. <title>Search Student by College Id</title>
112. </head>
113. <body>
114. <div class="search" align="center">
115. <form action="" method="post">
116. <p style="font-family:sans-serif">Student ID</p>
117. <input type="text" name="id"><br>
118. <input type="submit" class="submit" name="Submit">
119. </form>
120. {% with messages = get\_flashed\_messages() %}
121. {% if messages %}
122. {% for message in messages %}
123. <p>{{messages}}</p>
124. {% endfor %}
125. {% endif %}
126. {% endwith %}
127. {% if data %}
128. {% for key,value in result.items() %}
129. <table class="table-search" >
130. <tr>
131. <th>{{key}}</th>
132. <td>{{value}}</td>
133. </tr>
134. </table>
135. {% endfor %}
136. {% endif %}
137. <button class="submit" onclick="window.location='/'">Home</button>
138. </div>
139. </body>
140. </html>
141. **“~/templates/display-student.html” –**
142. <!DOCTYPE html>
143. <html>
144. <link rel="stylesheet" type="text/css" href="{{ url\_for('static', filename='style.css')}}">
145. <head>
146. <title>Display all Students</title>
147. </head>
148. <body>
149. <div id="list" align="center">
150. <table class="table-display" border="1">
151. <tr>
152. {% for i in keys %}
153. <th>{{i}}</th>
154. {% endfor %}
155. </tr>
156. {% for lines in result %}
157. <tr>
158. {% for value in lines %}
159. <td>{{value}}</td>
160. {% endfor %}
161. </tr>
162. {% endfor %}
163. </table>
164. <button class="submit" onclick="window.location='/'">Home</button>
165. </div>
166. </body>
167. </html>
168. **“~/static/style.css” –**

body{

background: #007fff;

}

#list{

background: #e6e6e6;

color: white;

font-family: sans-serif;

align-self: center;

width: 1000px;

margin: 100px auto;

border-radius: 10px;

box-shadow: 0 0 20px rgba(0,0,0,2);

}

.button{

background: white;

color: black;

font-family: 'Nunito',sans-serif;

font-size: 200%;

margin: 50px 20px;

padding: 10px;

text-align: center;

position: relative;

border-radius: 10px;

box-shadow: 0 0 5px rgba(0,0,0,0.5);

}

.button:hover,.submit:hover{

background: #004dff;

color: white;

cursor: pointer;

}

#entry{

font-family: sans-serif;

background: #e6e6e6;

color: black;

width: 300px;

align-self: center;

margin: 100px auto;

padding: 10px;

border-radius: 10px;

box-shadow: 0 0 20px rgba(0,0,0,2);

}

.submit{

background: white;

color: black;

font-family: 'Nunito',sans-serif;

font-size: 100%;

margin: 10px 50px;

padding: 10px;

text-align: center;

position: relative;

border-radius: 10px;

box-shadow: 0 0 5px rgba(0,0,0,0.5);

}

.search{

font-family: sans-serif;

background: #e6e6e6;

color: black;

padding: 20px;

width: 500px;

margin: 100px auto;

border-radius: 10px;

box-shadow: 0 0 20px rgba(0,0,0,2);

}

.table-search{

width: 400px;

overflow: hidden;

}

.table-search th{

background-color: #007fff;

color: white;

text-align: left;

font-weight: bold;

}

.table-search th,.table-search td{

padding: 12px 15px;

width:16.6%;

word-break: break-all;

border: 1px solid black;

}

.table-search tbody tr{

border-bottom: 1px solid #e6e6e6;

}

.table-search tbody tr:last-of-type {

border-bottom: 2px solid #007fff;

}

.table-display{

border-collapse: collapse;

color: black;

margin: 25px 0;

font-size: 90%;

min-width: 400px;

border-radius: 10px 10px 0 0;

overflow: hidden;

box-shadow: 0 0 20px rgba(0, 0, 0, 0.15);

}

.table-display th{

background-color: #007fff;

color: #ffffff;

text-align: left;

font-weight: bold;

}

.table-display th,

.table-display td {

padding: 12px 15px;

}

.table-display tbody tr {

border-bottom: 1px solid #dddddd;

}

.table-display tbody tr:nth-of-type(even) {

background-color: #f3f3f3;

}

.table-display tbody tr:last-of-type {

border-bottom: 5px solid #007fff;

}

1. **Approach –**

The approach towards this problem statement was fairly simple. The CSV module is used to read and write the ‘colleges.csv’ file. The user was given three options which were Registering a new college, Searching a college and Deleting a college from the csv file. So first the user’s input was considered for the above options and depending on the input, the program flow was redirected to the specified functions which performed the desired task. A new college is stored in an object of the College class with it’s different attributes and method. This was done solely on the Python console to run and compile the commands for the csv file editing. When everything was done, I started building the GUI using the “Tkinter” module. The tkinter module is very useful and efficient for designing python applications. The user is given three options with the three buttons and depending on which button is pressed, the respective data entry options appear. For example if user chooses to search a college, after the button click, the window will display entry boxes so that user can provide the parameters such as College Name and Course type.

1. **Coding Practices –**

The use of CSV and Tkinter modules makes the code more efficient than other programming languages. Lambda function was also used. Lambda is a small anonymous function which contains only single expression. It is mainly used to write functions which are going to be used only once. In this code, it was used to pass the user entries as formal parameters to the called function. This made the code more concise and short. The use of different methods made the code very modular. So even if one part of the code malfunctions, it won’t affect the whole program.

1. **Screenshots –**

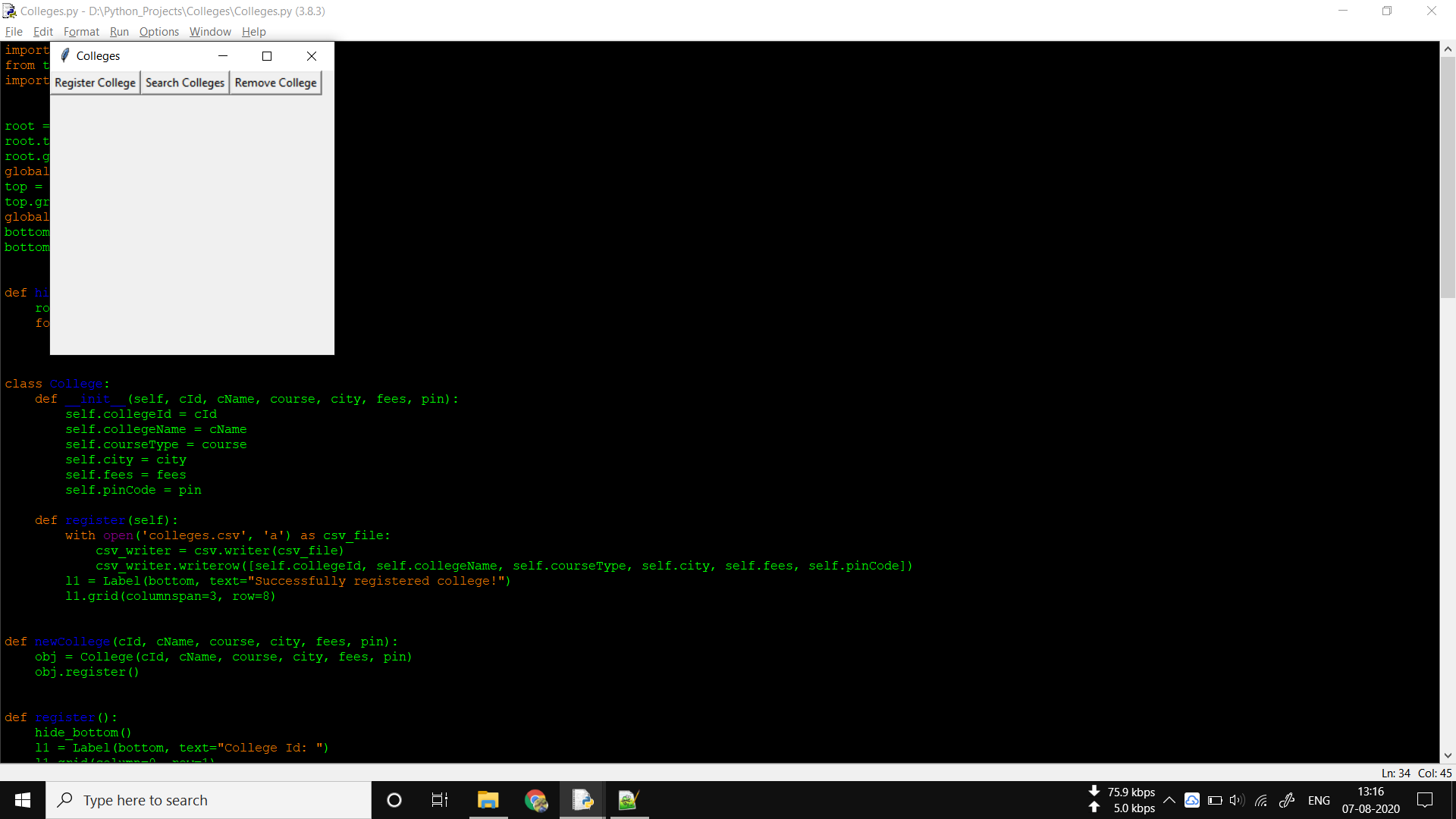


Fig1 – Options given to the user in the application

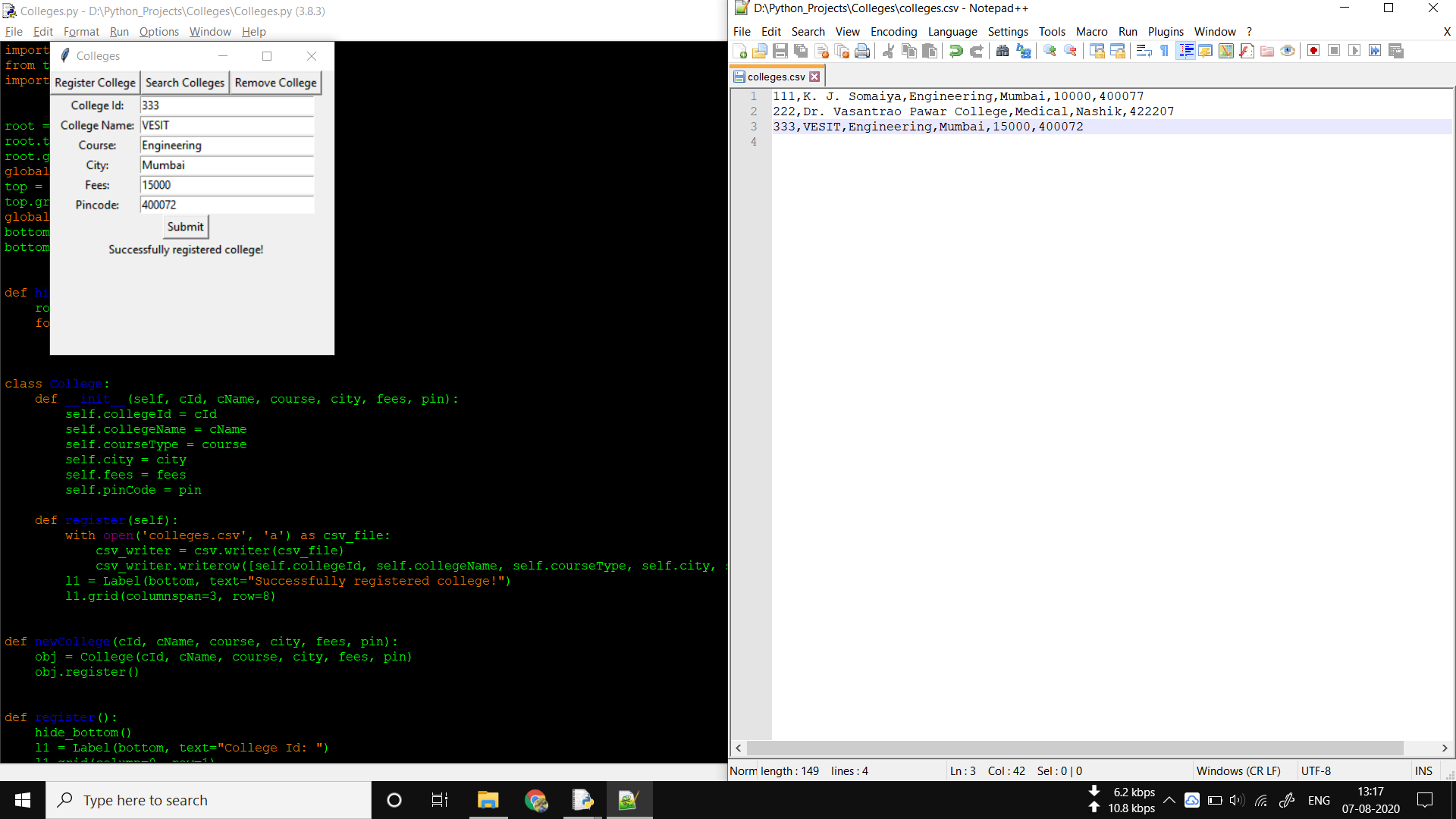


Fig2 – Registering a new College

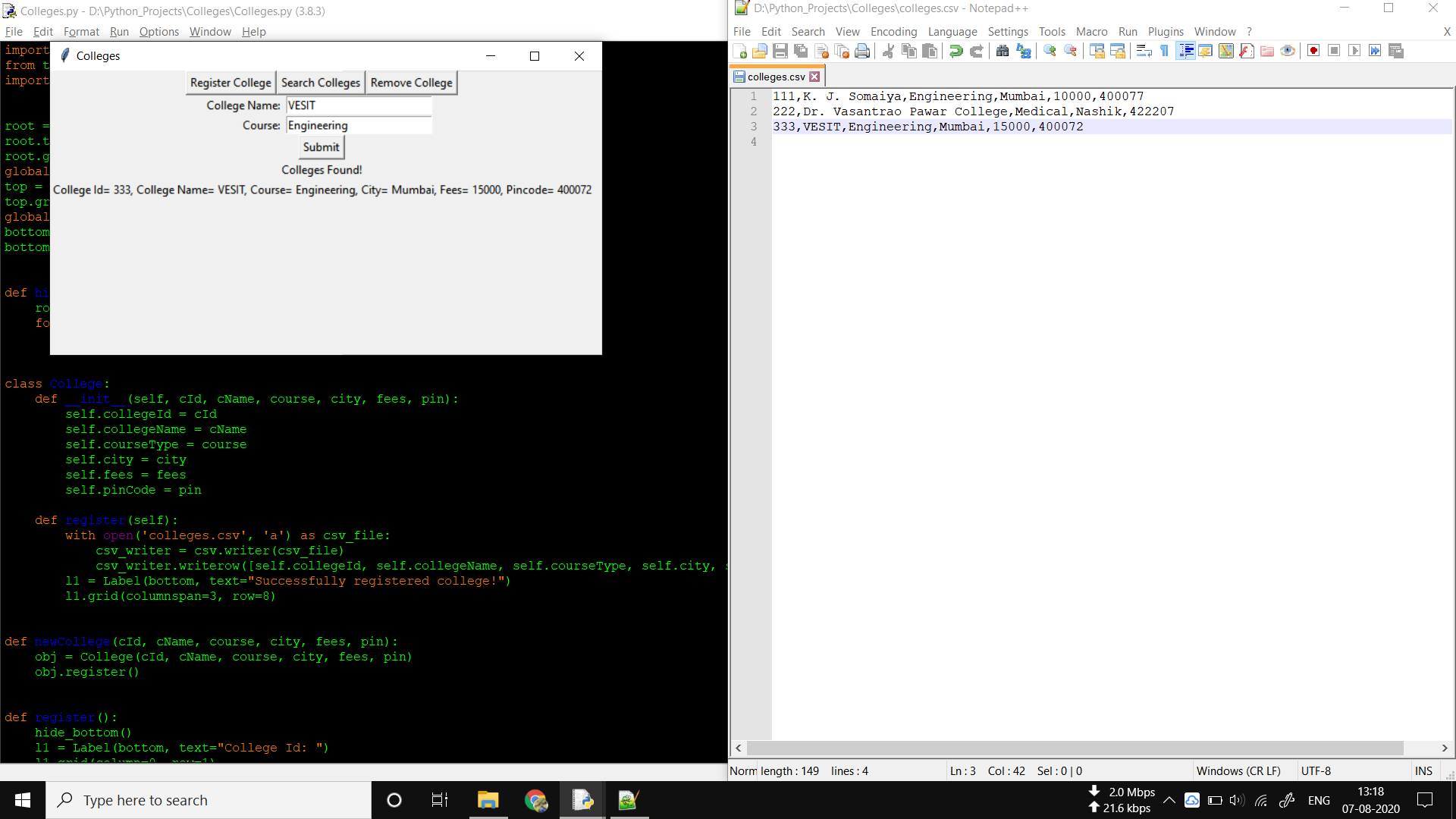


Fig3 – Searching a College

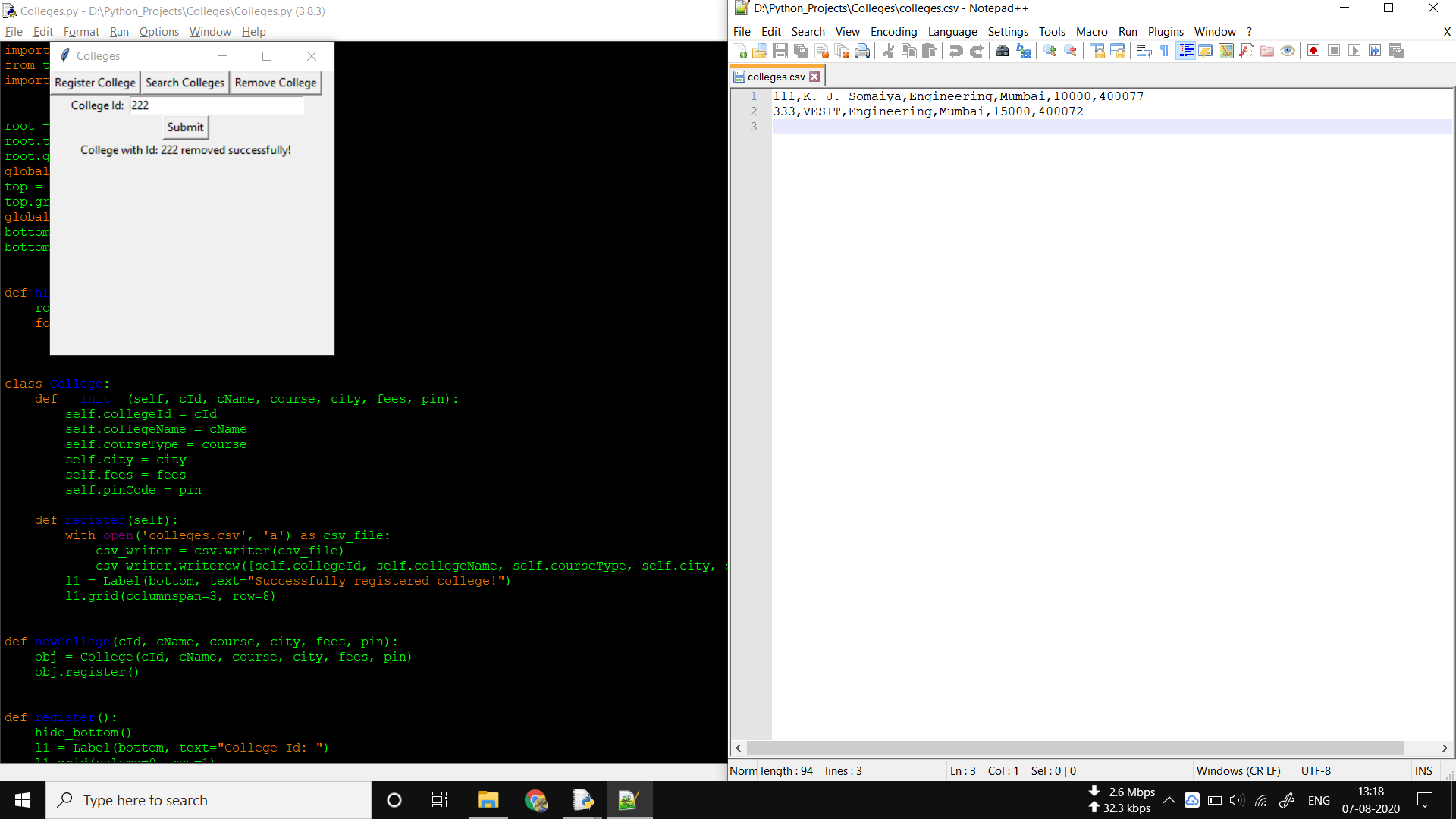


Fig4 – Removing a College