**Criterion C: Development**

**Word Count: 1011**

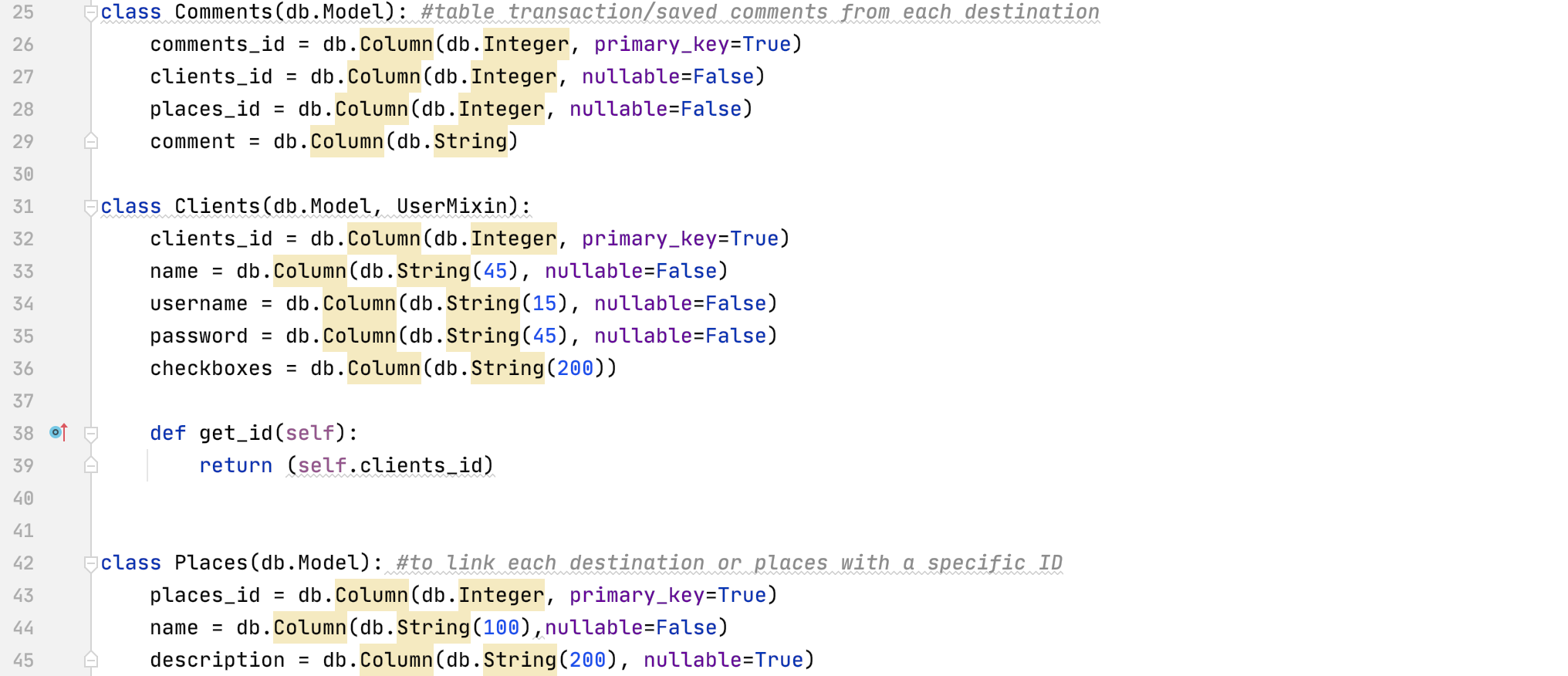
***List of Techniques***

The techniques used to create this program include:

* Variables
* Methods
* Data validation
* Exception handling
* Arrays
* Looping techniques
* Sorting
* Miscellaneous (HTML attributes)

***Variables***

All variables in the three main database classes are public variables.



*Fig 1.1 - Three classes referring to database tables*

To display comments in particular places made specifically by the client, the Comments class has a primary key (comments\_id) and two foreign keys (clients\_id from Clients table and places\_id from Places table). Both foreign keys have a one-to-many relationship, indicated in the UML diagram at Criterion B, as one user can have many comments and can comment at many places.

***Methods***

With Python as back-end and HTML as front-end language, Flask is vital for the web framework. Throughout my code, I use two basic HTTP methods for data transfer between server-side and client-side: GET and POST. The Get method requests unsecured data from the server, whereas the Post method sends encrypted data to server to process data.

For example, the comment1 function uses Get method and Flask’s query method to display all the client’s comments about Shaoxing Restaurant.



*Fig 1.2 - The static function that displays the client’s comments about Shaoxing Restaurant*

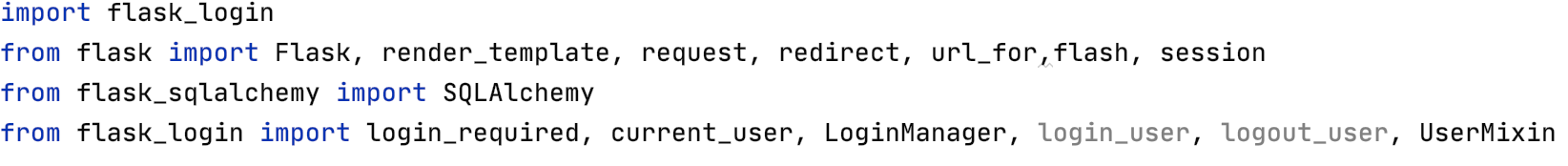


*Fig 1.3 - Process of querying the client’s comments to be displayed (server-side processing[[1]](#footnote-0))*

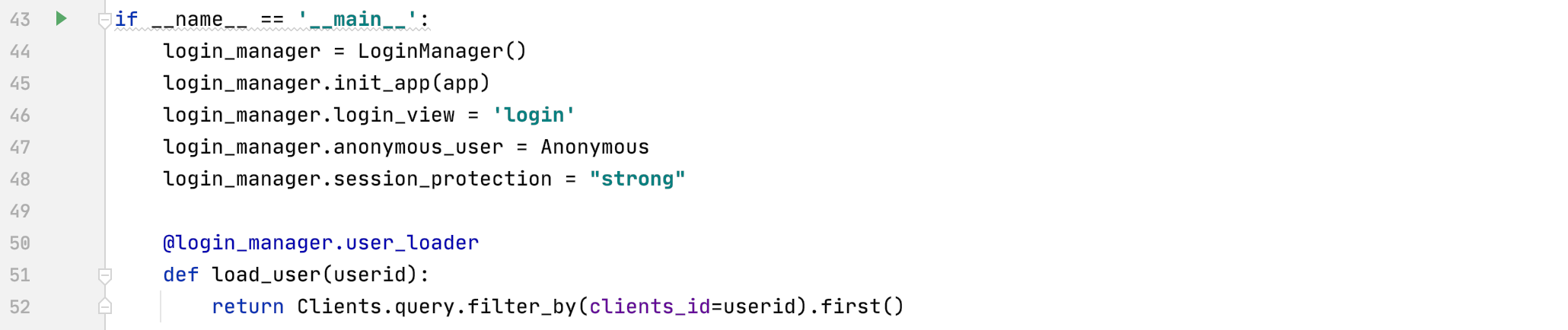
The process of saving comments to the database and displaying previous ones (Fig1.3) uses Post method and Flask’s request, add, commit and query method. These are the program’s main functions and operations in which clients can save and review their comments.

***Data validation***

Since clients can hold their own information on places visited and comments about their trip, I created the login and register function. First, I imported various Flask libraries that will act as function templates.

****

*Fig 1.4 - Flask’s import template modules*

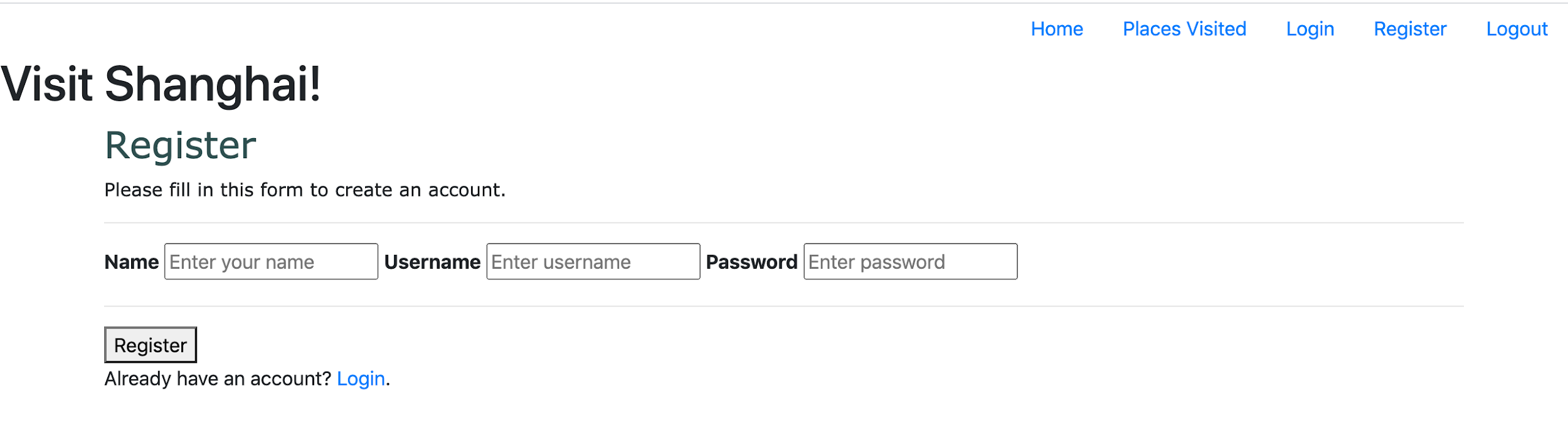


*Fig 1.5 - Login manager system for the program to work with Flask-login*

The LoginManager class allows *Flask\_login* mechanismto work and uses the *login\_view* configuration to direct clients to the ‘login’ page when initially accessing the website (line 44-46).

A *user\_loader* callback reloads current user in session and generally ensures client can access their own accounts when logged-in.

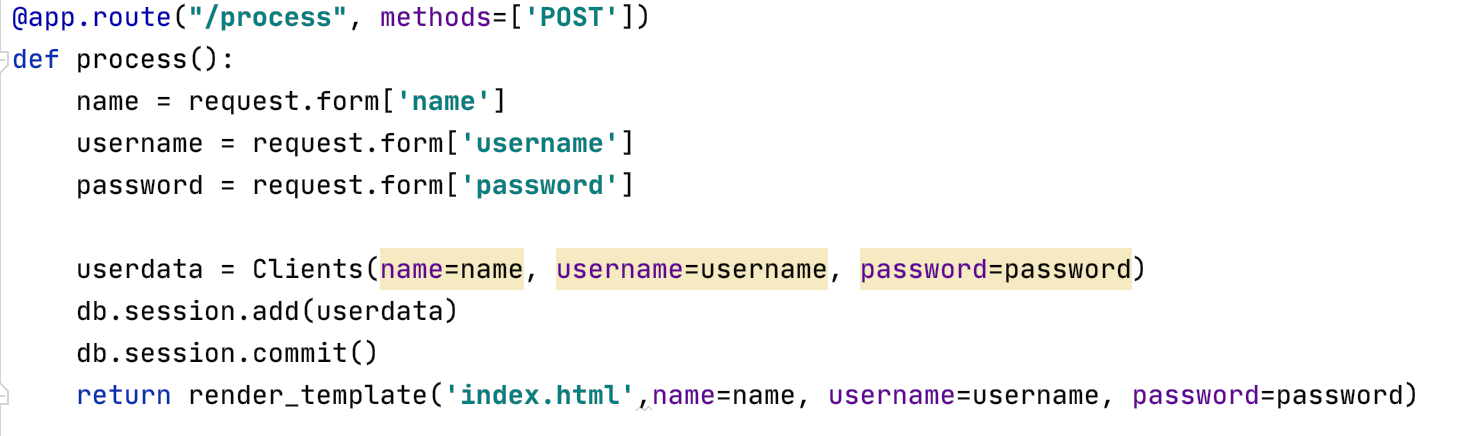
If client is inactive for a long time, the program will automatically redirect to the login page and session protection becomes ineffective (line48).



*Fig 1.6 - Register HTML page.*



*Fig 1.7 - Register HTML code*

All HTML pages, like Fig1.7, start and end by overriding the parent template (‘base.html’), using Jinja’s template block and endblock tags (lines 1, 30) to implement the menu bar. 

*Fig 1.8 - Register processing function (server-side processing)*

After clicking ‘Submit’ on the register page, the process function’s Post method adds user input into Clients table. It uses Flask’s *request.form* attribute as dictionary object to process data sent to server and match data to their specific database columns through *userdata* variable(“Flask – Request Object”)*.* All data are saved using Flask-SQLAlchemy records function *db.session.add* and *db.session.commit.*



*Fig 1.9 - Login function to check user’s inputted username and password with data in database*

Similarly, login function’s Post method requests client’s username and password from server (line 65-66) and queries by filtering these two data. An if-else statement verifies whether client’s inputted data matches those in the database. *Flask\_login* mechanism (line71) checks if data exists.



*Fig 1.10 - Login HTML code*

In Fig1.10, the username and password refers to the *username* and *password* variable (line 21,24) in the login function (Fig1.9,line 65-66). An if-endif statement shows any error *message* when logging-in (line 27-30). This often occurs after client’s initial registration.

***Exception Handling***

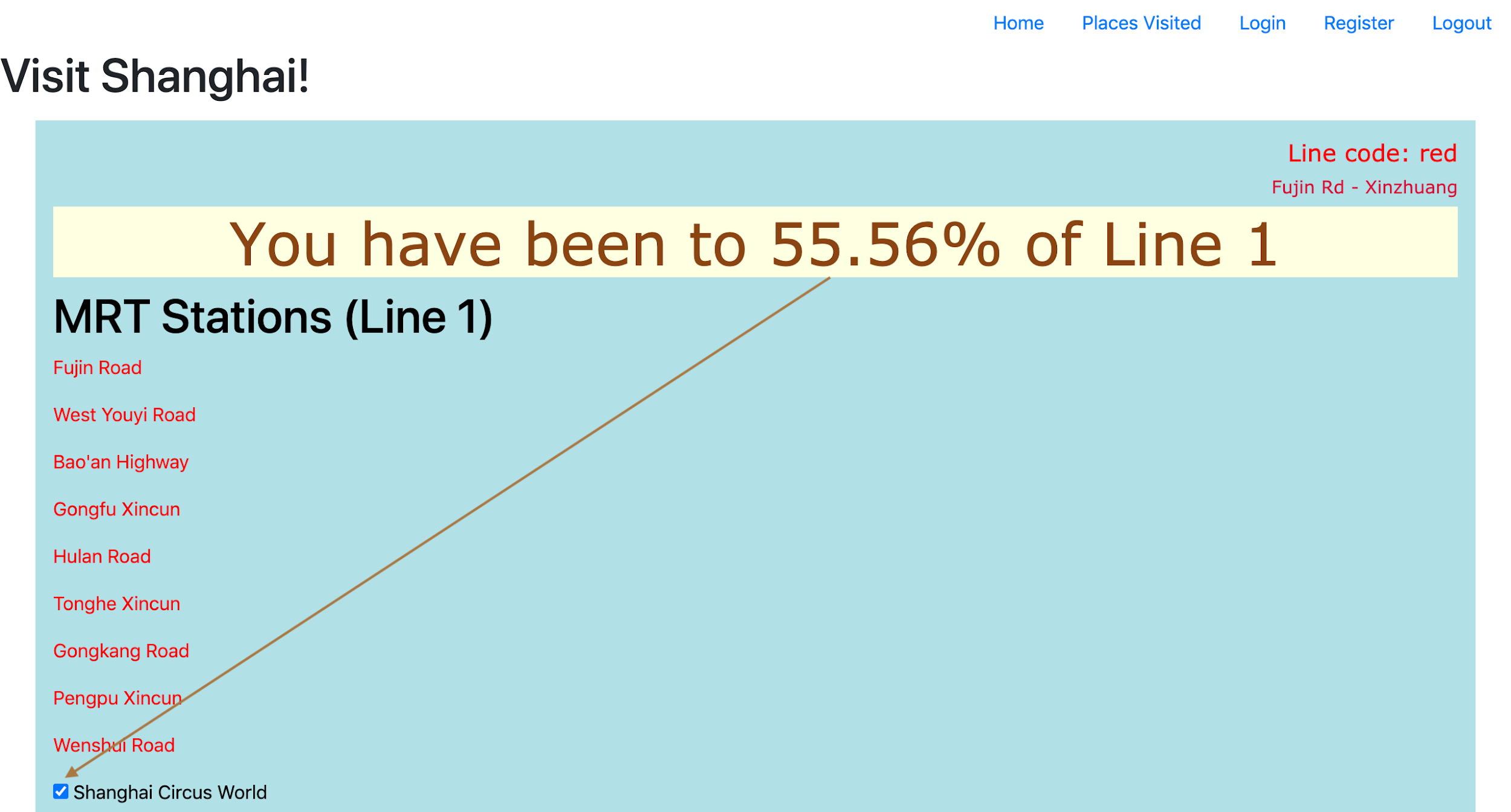
The *line1* function calculates the total percentage of places visited by the client. Since the client may not have visited all stations, an error can occur during the calculation. Hence, a try-except statement fixes that problem. If client ticks a checkbox, the *checkboxes* value becomes an array and the total percentage of checked checkboxes is calculated in *jumlah[[2]](#footnote-1)* variable. The mod operation (line127) simplifies the percentage to two decimal points.

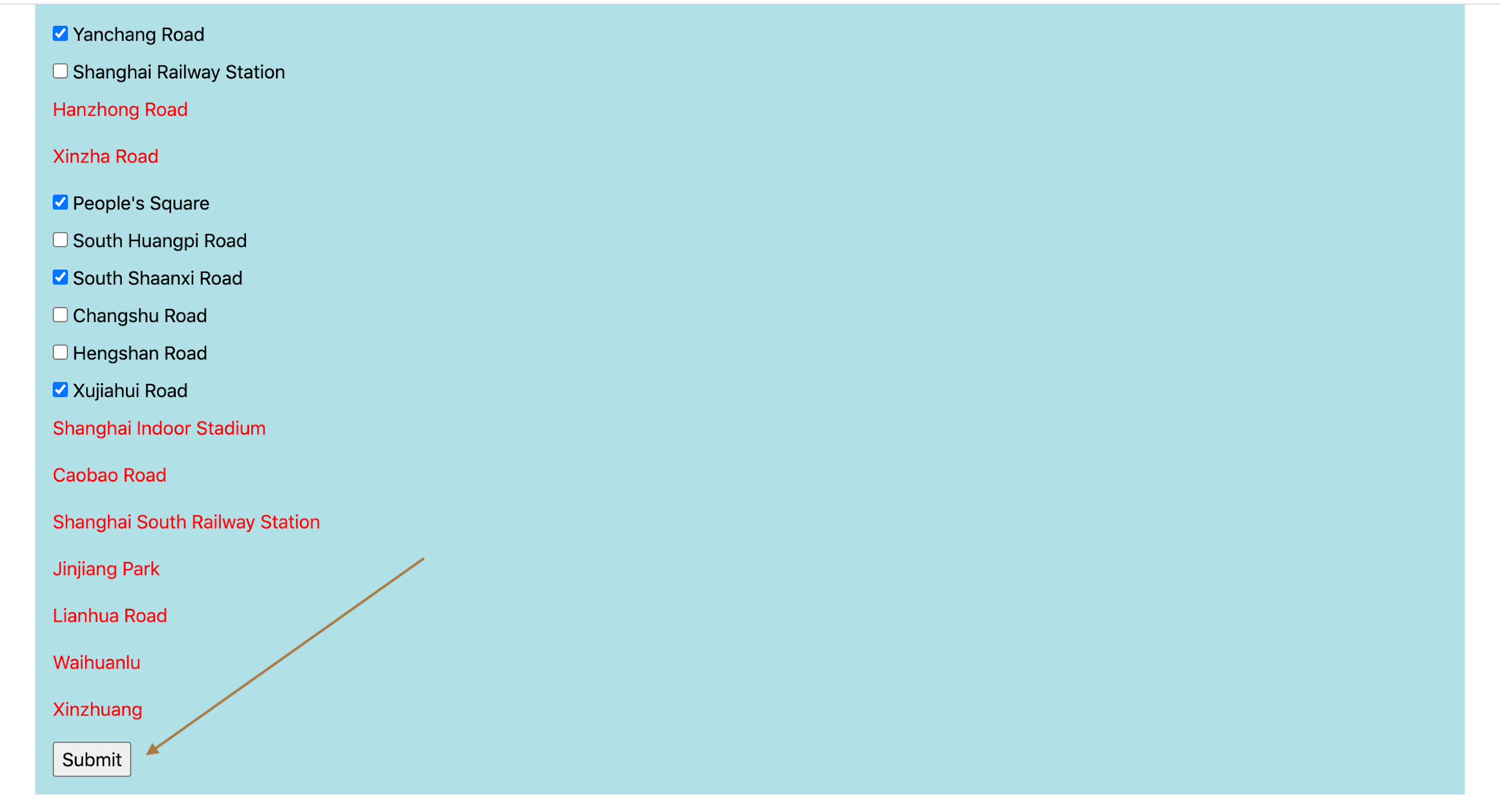


*Fig 1.11 - Line 1 function to calculate and display the percentage of checked checkboxes*

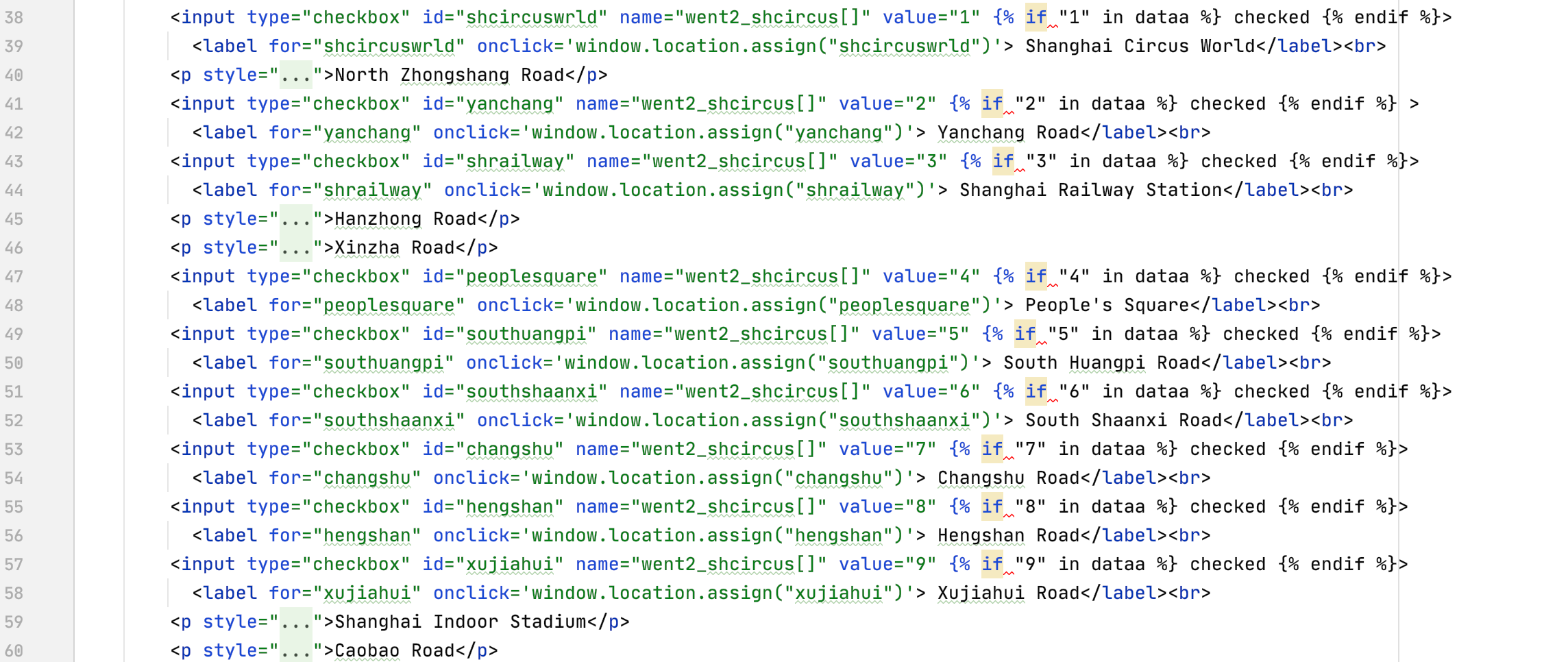
This function’s processing (line1process function) also has similar try-except handling (Fig 1.14).

***Arrays***

****

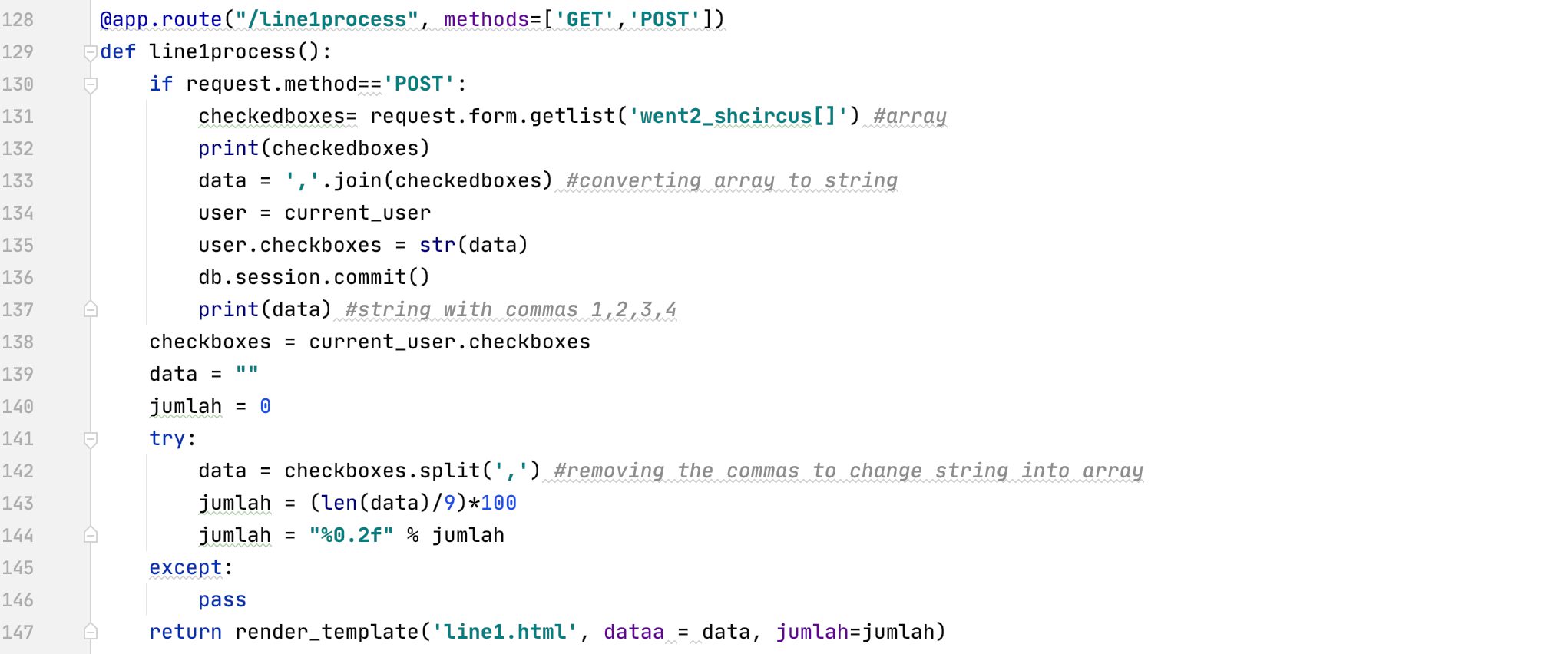


*Fig 1.12 - Line 1 HTML page*



*Fig 1.13 - Line 1 HTML code with if-statements*

To save client's checked (visited) stations, data structures are converted from arrays to strings. Initially, each checked value is saved into *went2\_shcircus[]* array. If-endif tags then loop around the ‘*dataa’* string of checked checkboxes.



*Fig 1.14 - Line 1 processing function to convert arrays into strings, and vice versa for percentage calculation (server-side processing)*

*Request.form.getlist* method declares the array to *checkedboxes* variable. Lines 130-137 convert the array to string and save it in the *checkboxes* column in table Clients using *db.session.commit().* Generally, to avoid confusion, I decided to distinguish which data sets will be saved in the database (strings) and which will be used for calculation (arrays).

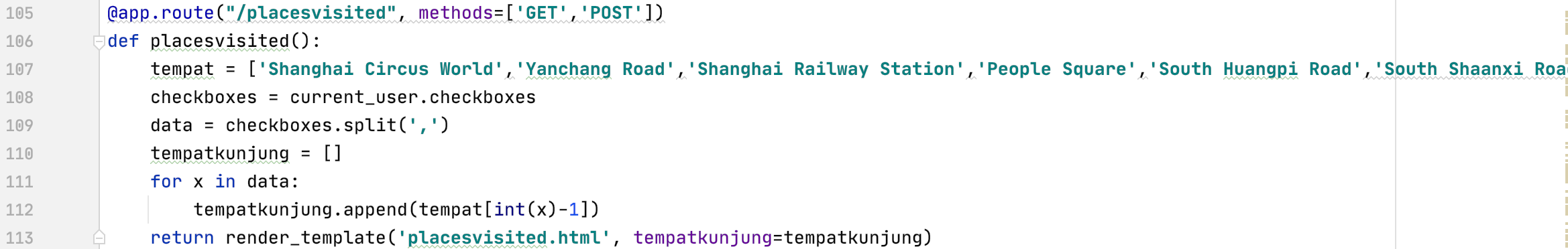
***Looping Techniques***

For-loops are used to continuously repeat processes until it reaches the desired output.



*Fig 1.15 - Looping in the front-end to display an error or message*

In most cases, it is used in HTML using Jinja template tags to get and display messages, either error or success statements (lines 144-146 & 149-151). A continuation in using for-loops can be found in the *Sorting* chapter.



*Fig 1.16 - Displaying a list of places visited by the client (checked checkboxes)*

Another example used is in *placesvisited* function, which displays an array of places (MRT stations) that clients have visited. After changing the checked checkboxes string to array (line 109), each place in the *data* variable is repeated in a for-loop (line 111), added to an empty array *tempatkunjung[[3]](#footnote-2)* using Python’s .*append* method, and returned.

***Sorting***

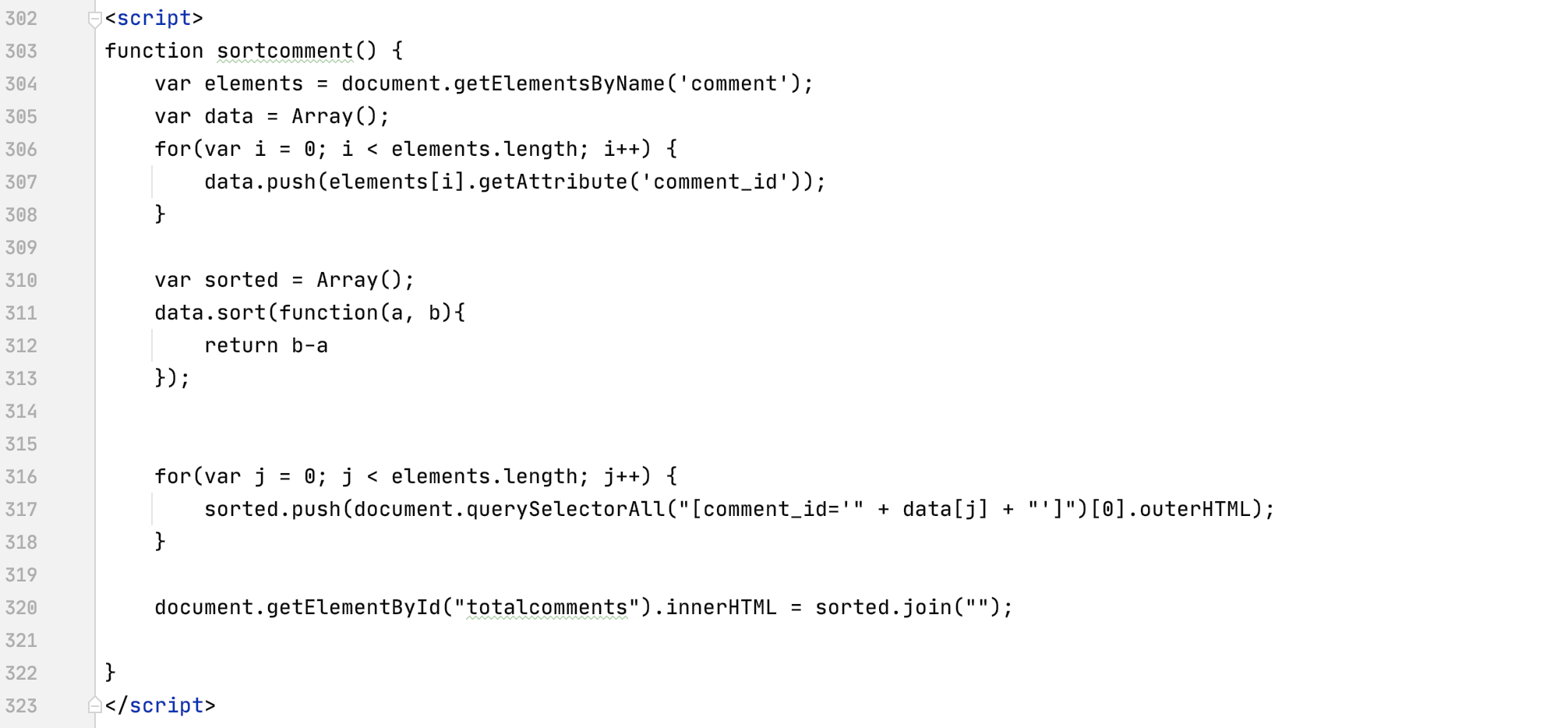
I use bubble sorting algorithm to sort, the most to least recent, client’s comment. As it is rather difficult to link a Python function with an HTML button, I decided to code the sorting algorithm using Javascript altogether.



*Fig 1.17 - HTML page (include sorting button) to sort out the list of comments made by the client*

First, I assign the client’s comments with an id (line 239), so that the JS function can refer to the comments. <li> tags puts all the comments into a list (line 242) to easily convert it into an array for sorting.

The function, *sortcomment(),* converts the list to an array and does bubble sort.



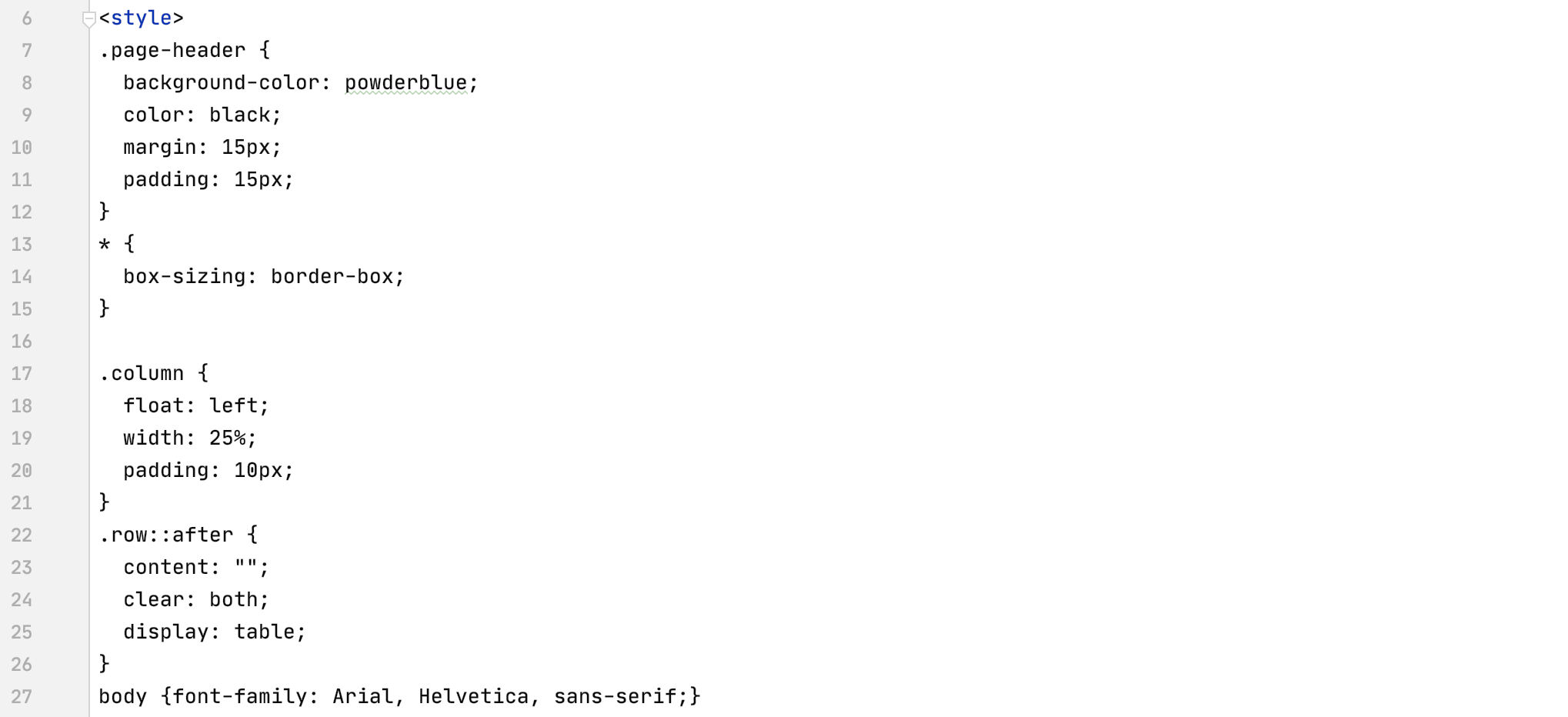
*Fig 1.18 - Javascript function to change list into array and do bubble sort*

A local variable ‘*elements’* gets all the comments’ comment\_id and puts them into an empty array using for-loop (lines 304-307). In lines 310-312, I use Javascript’s sort method and pass the *compareFunction* parameter to compare each value in the array. It returns a positive value, displaying a greater value first following a smaller one (descending order).

For-loop adds all sorted data into the *sorted* variable, by using the push and querySelectorAll() method to return all data, and HTML DOM outerHTML property to sort without having to refresh the page. Finally, the innerHTML property changes the HTML element *‘totalcomments’* to the sorted list of comments.

***Miscellaneous (HTML attributes)***

For client to have a brief background about each destination, several HTML elements, like class attributes, are used to display pictures about each place.

I first create classes *page-header, column,* and *row.*  

*Fig 1.19 - HTML styles when displaying pictures of each destinations*

Then, I use <div> element to call each class and retrieve particular pictures from *static* folder*.*



*Fig 1.20 - HTML attributes to display pictures*

Bibliography

“Flask – Request Object.” *Tutorialspoint*, www.tutorialspoint.com/flask/flask\_request\_object.htm#:~:text=Flask%20%E2%80%93%20Request%20Object&text=Form%20%E2%88%92%20It%20is%20a%20dictionary,holding%20Cookie%20names%20and%20values.

1. Functions that are written as ‘server-side processing’ cannot be directly accessed when typing into the webpage. These functions must go through the main function (static functions) first before reaching this processing page. [↑](#footnote-ref-0)
2. *Jumlah* in Indonesian means total. [↑](#footnote-ref-1)
3. *Tempatkunjung* in Indonesian means placesvisited. Due to the use of another “placevisited” variable in the program, I use an Indonesian version of it. [↑](#footnote-ref-2)