首先我们用到了保存全局变量的g属性：

g: global

g对象解释： 就是为了保存用户一些自定义参数

g对象是专用用来保存用户的数据的。

g对象在一次请求中，全局可以调用。

First, we use the properties of the global variable g:

The variable g is used to avoid modification of self-defined variables used by the user.

It is also used to protect the data collected by the user. Another important property of the global variable g is that it can be used globally within one request.

The application context is a good place to store common data during a request or CLI command. Flask provides the [**g object**](https://flask.palletsprojects.com/en/2.0.x/api/#flask.g) for this purpose. It is a simple namespace object that has the same lifetime as an application context.

The g name stands for “global”, but that is referring to the data being global within a context. The data on g is lost after the context ends, and it is not an appropriate place to store data between requests. Use the [**session**](https://flask.palletsprojects.com/en/2.0.x/api/#flask.session) or a database to store data across requests.

A common use for [**g**](https://flask.palletsprojects.com/en/2.0.x/api/#flask.g) is to manage resources during a request.

* g lives in the request context, i.e., created afresh when the requests starts, and available until it ends
* g is intended to be used as a "request blackboard", where I can put stuff relevant for the duration of the request (i.e., set a flag at the beginning of the request and handle it at the end, possibly from a before\_request/after\_request pair)
* in addition to holding request-level-state, g can and should be used for resource management, i.e., holding database connections, etc.

1. Separate code blocks and explanations for each of the python functions

We write three python functions for database management in a file app.py.

1. We define get\_message\_db () to handle creating the database of messages.
2. Firstly, we check whether there is a database called message\_db in the g attribute of the app. If not, then connect to that database.

(2) We check whether a table called messages exists in message\_db, if there is no table available, we create the table messages with three columns: ID as an integer, handle as text, and message as text.

(3)return the connection g.message\_db.

1. The second function we created is called insert\_message() which should handle inserting a user message into the database of messages.

(1) We extract the message and the handle from request if they exist. We should ensure submit.html template creates these fields from user input.

(2) Then we connect the database using cursor.

(3) To ensure that the ID number of each message is unique, we set the ID number equal to one plus the current number of rows.

(4) Write a SQL command in a string called g.sql with variable input value, then perform the insertion with cursor.exectute(g.sql).

(5) Run db.commit() to ensure the row insertion has been saved.

1. The third function we created is random\_messages(n), which fetches n random messages from the message\_db.
2. We connect the database using cursor.
3. Execute sql command to extract the message from message database in random order.
4. Get n rows from the result.
5. Close the cursor and the database.
6. Return the result.
7. This is the fourth function I created. The function is straightforward, as it gives us the main page of the Webapp.

**I put navigation links(insert and view links at the top of the screen) inside a template called base.html provided by the professor. Then I had the main.html, submit.html, insert.html templates extend base.html.**

1. The fifth function we created is called view (). The function creates a viewable webpage.
2. Call the template view.html and fetch the number of messages from the data we inputted.
3. We will fetch a function called random\_messges(n) from db\_app.py to pick n messages from the database randomly.
4. If we submitted nothing, the program outputs error.
5. The last function we defined is called submit ().
6. From template submit.html, we extract handle and message from user input.
7. If they exist, we will fetch a function called insert\_message from db\_app.py.
8. The insert\_message () function will insert a user message into the database of messages.
9. Discussion of a template file I created in my app

I choose to discuss the template file view.html. The file view.html presents us with the random messages chosen if we click on a button.

1. First, we display the line of text “please input how many messages do you want to view (1-5)?” on the website.
2. Then we design the input box so that it only accepts numerical input and the value ranges from 1 to 5.
3. When we press the submit button, the program will use the function random\_messages(n) from app.py to return n messages from our “messages” list (the number n is inputted by the user) and record it into g. result.
4. return render\_template('view.html')返回到view.html, 在页面的下端显示n条记录，方法是做一个循环，依次取出g.result中的每一条记录送入r，r[0]是message，r[1]是handle，用斜体显示。

We take advantage of the fact the Jinja tags support looping and indexing of objects, so we looped over the messages.

If r is a tuple of user handle and message, r[0]contains the message and r[1] contains the handle.

1. If there is an error, we return the error message.

All in all, we create a header, tell the user to submit some interesting message, and finish the html by giving some font and size.