This is a brief rewrite of the *session* function from the original Flask code that was created for an internal dev team. They wanted friendlier, more comprehensive Flask documentation in order to help less experienced developers get up to speed more quickly.

This section was improved by:

- Expanding the explanation of what a *session* is and how it can be used to store data.
- Adding comments to the code for clarity.
- Adding white space and extra lines for ease of reading.
- Placing notes at the end of the section to alert the reader of certain default limitations in the session object, such as the maximum data storage limit and the permanent session lifetime setting.
- A mention of the escape() function was also included to make the reader aware of the possible security implications of displaying unescaped data in the browser.

Summary: This section briefly outlines the use and function of the session object. It covers the basic concept of a session along with examples of situations where session functionality might be useful (for example, remembering the login state or the contents of a shopping cart). We also briefly cover the inherent statelessness of the HTTP protocol to show what functionality the session object adds that HTTP lacks by design. Finally, a simple code example is provided showing you how to use the session object to store and recover session variables.

Sessions

What is a session?
A *session* is a simple, secure method used to store and remember information about a specific user as they navigate from page to page across a website.

Why do we need sessions?
Each time your browser requests a page from the server, the server fulfills the request using the HTTP protocol. After the page has been sent the server forgets everything about the request. This is what is referred to as a *stateless* protocol, meaning that no information about the previous request (the state) is remembered.

By using the `session` object you can tell the server to remember specific information about a user. Examples of this might be whether or not the user is logged in or the contents of their shopping cart. You can think of a session as a bit of private "scratchpad" memory set aside just for your use.

How are sessions accessed?
Sessions require a key that is used to cryptographically sign the session cookie. Depending on the session

type, this cookie is a key that either holds or identifies a specific session's data. The following example demonstrates how sessions work:

```
# note that we need to import 'session' explicitly just like we do for
# redirect, url_for, escape, and request
from flask import Flask, session, redirect, url_for, escape, request

app = Flask(__name__)

@app.route('/')
def index():

    # our first use of the session object is to
    # check to see if it contains a username
    if 'username' in session:

    # if it does, tell the user that they're logged in
        return 'Logged in as %s' % escape(session['username'])

    # else show the "not logged in" message
    return 'You are not logged in'

# accept both GET and POST form methods
@app.route('/login', methods=['GET', 'POST'])
def login():

    # if using POST, we'll set the session var 'username'
    # to whatever the user has entered in the form field
    if request.method == 'POST':
        session['username'] = request.form['username']
        return redirect(url_for('index'))

    # if the user is not logged in, display the login
    # form with the username and password fields
    return '''
        <form action="" method="post">
            <p><input type=text name=username>
            <p><input type=submit value=Login>
        </form>
    '''

@app.route('/logout')
def logout():
    # delete the username from the session if present,
    # then send the user back to the index page
    session.pop('username', None)
    return redirect(url_for('index'))

# set the encryption key. Use a long key (32 chars or more)
# and store it securely (e.g. in a database or protected file)
app.secret_key = 'Aw#g%rdj)*Om?47F[39K@:4&b^8u43q5'
```

Additional Notes:

1. The default Flask session uses a *client-side* implementation, meaning that Flask will take the values you put into the session object and serialize them into a cookie. Cookie size is limited to 4096 bytes in most browsers and data in excess of that limit will be discarded. For storing session data in excess of 4096 bytes you will need to enable *server-side* sessions in Flask. When using server-side sessions the cookie itself does not contain the stored data; instead it contains a session ID encoded with the `secret_key` that links to the stored data on the server.

2. This example doesn't make use of the template engine, so the `escape()` function is used to make sure that the user name data is safe for display in the browser. If this isn't done then it's possible that a malicious user could use a specially-crafted name to perform XSS exploits, load Javascript code, or perform other harmful behaviors.

3. Default ("volatile") sessions in Flask automatically expire when the browser is closed. If you have set sessions to be permanent then the session will persist for the number of seconds specified in the `permanent_session_lifetime` configuration setting. The default is 31 days (2,678,400 seconds).

4. More detailed information on the session object, including object properties and configuration settings can be found at the following URL: http://flask.pocoo.org/docs/0.10/api/#flask.session