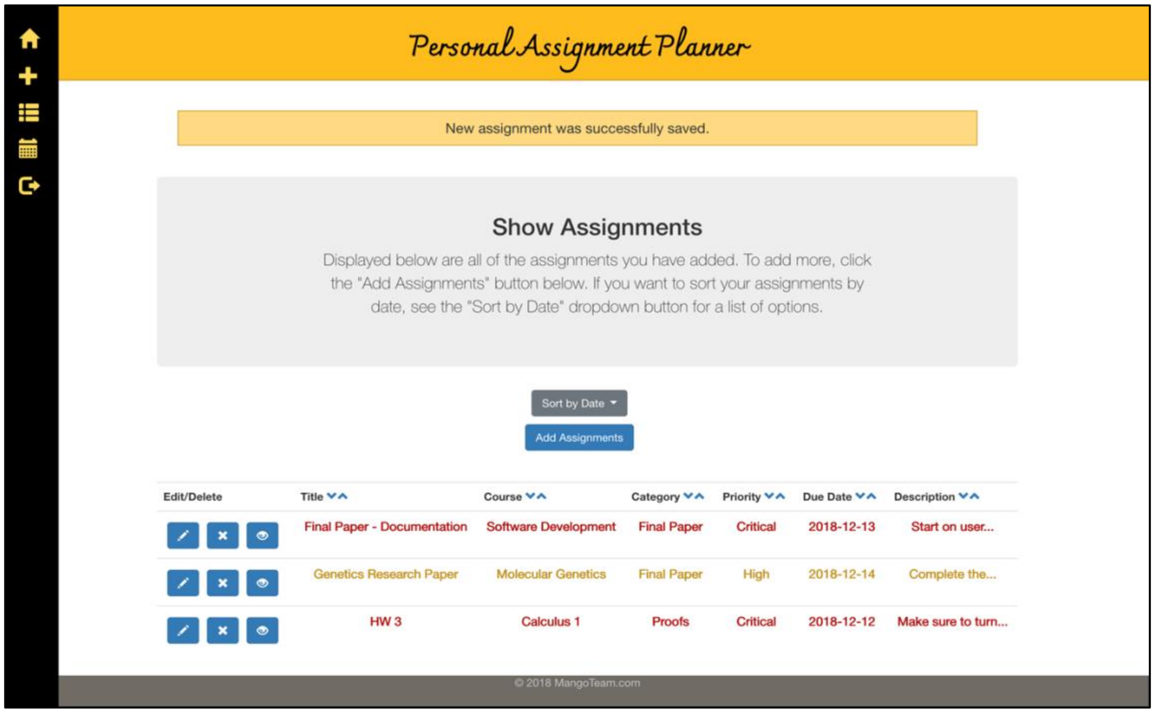
ASSIGNMENT PLANNER

Application Documentation



New assignment was successfully saved.

Show Assignments

Displayed below are all of the assignments you have added. To add more, click the "Add Assignments" button below. If you want to sort your assignments by date, see the "Sort by Date" dropdown button for a list of options.

Edit/Delete	Title	Course	Category	Priority	Due Date	Description
	Final Paper - Documentation	Software Development	Final Paper	Critical	2018-12-13	Start on user...
	Genetics Research Paper	Molecular Genetics	Final Paper	High	2018-12-14	Complete the...
	HW 3	Calculus 1	Proofs	Critical	2018-12-12	Make sure to turn...

© 2018 MangoTeam.com

Megan Dolan

December 13th, 2018

TABLE OF CONTENTS

USER’S PERSPECTIVE

The purpose of the application is for users to add and organize their assignments based on features that they can control. Some of these features include course names, assigned priority level, and even personalized assignment categories.

SESSION FEATURE

Users are able to create an account, login, and logout as needed. This is primarily for user access and security to the user’s individual assignments. When users log onto the opening page, there are two buttons asking the user to create an account or log into an existing account. The user can then select which option. If the user creates an account, they will be asked for a username, for a password, and to repeat their password. Then, once the user has completed those fields, they can then log in. However, if the passwords are incorrect, the user will then need to enter in the fields again and will not be granted access to create an account. Each of these fields are also required, so a user cannot skip over an input. The user may also face an error if they try to create an account which already exists. This error prevents multiple users from having the same user information.

If a user chooses to log in first, they are only prompted for a username and password. If both the username and password match an existing account, then the user will be logged in and redirected to their homepage, or dashboard. If the username exists, but the password does not match, a flash message will show up alerting the user as such. Likewise, if a user enters in a username that doesn’t exist, then a flash message will pop up indicating that as well.

DASHBOARD FEATURE

By clicking on the icon in the side navbar that looks like a home as indicated in Image 1, it will take you to the user’s main dashboard. The user will also be taken to the homepage when they are first logged in. If a user attempts to go back to the opening page, as written in `OpeningPage.html`, then they will be redirected back to their homepage in order to prevent the user from signing in without logging out first.

The homepage acts as a dashboard for the user, showing them the number of assignments and other information pertaining to the user’s profile. One of the features of this page is that it lists the number of assignments the user has for each priority. There are four different priority types: critical, high, normal, and low. This acts as a system to a user, which represents the priority of the assignment to help the user plan their schedules. The default option for the priority is normal, so this only changes the assignments as it is indicated. The homepage also displays several of the most recently added assignments. There is a link below redirecting the user to the show assignments page to view the rest of the assignments.

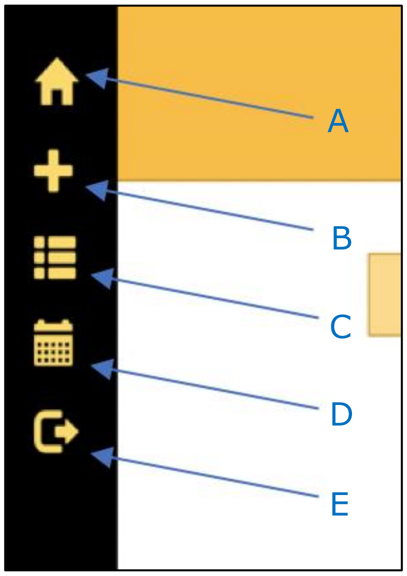


Image 1. Side navbar with icons indicating the individual pages. A) dashboard/homepage, B) Add assignment, C) Show assignments, D) Calendar, and E) Logout.

ADD ASSIGNMENTS FEATURE

The main purpose of the software is based off of the ability to add assignments as needed. If a user clicks on the "+" icon in the side navbar, then they will be redirected to the "add assignments" page (Image 1). Here, the website displays a form with several inputs. These inputs include the title of the assignment, the course that the assignment was assigned in, a "category" option in case a user wants to categorize the assignments a certain way, a dropdown menu of four different priority options, a due date option, and a description field where a user can include any other important details. The only required inputs are the priority level and the title of the assignment. However, the priority is automatically selected for "Normal," therefore a user does not have to manually select at least one priority level.

Once a user inputs the information regarding the individual assignment, they can then submit it. When the information is then submitted, the program stores the assignment with a hidden attribute to the user's account. That way, any added assignment is associated with the user that created it. This becomes important when a user views their assignments.

SHOW ASSIGNMENTS FEATURE

Another primary feature of the application is being able to view the assignments associated with each user. On the side navbar, there is an icon that looks like a list. If a user clicks on this icon, then they will be routed to the “show assignments” page (Image 1). On this page, the user will see a list of all of their assignments. Above the list of assignments, there are two buttons. One of these buttons, labeled “Add Assignments,” acts as a link to the add assignments page. The other, the “Sort by Date” dropdown button, displays assignments based on the duedate selected via the dropdown menu (Image 2). Only dates corresponding to assignments added by the user will appear, so the user only has the option to sort their assignments by corresponding dates.

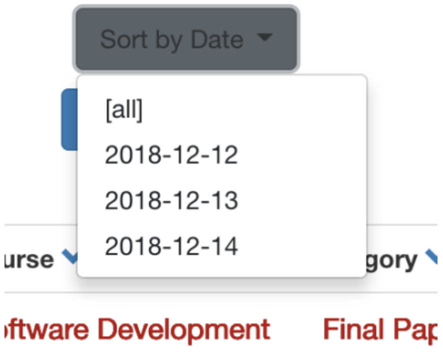


Image 2. Sort by Date dropdown button.

On the show assignments page where the assignments are displayed, a user can also sort the assignment by any of the corresponding columns. Each column has two arrows, and by simply clicking on the up and down arrows next to each column name, the assignments will be sorted by that attribute (see Image 3). By clicking the up arrow, this will sort the assignments by ascending order. Similarly, the down arrow sorts the assignments in descending order.

Having this option gives the user the freedom to organize their assignments by multiple variables.

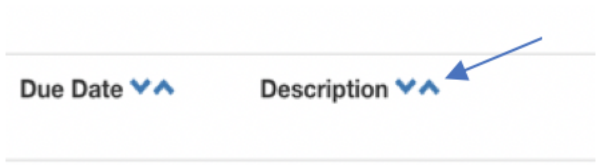


Image 3. Ascending and descending sort feature based on

column name.

One of the main objectives is to create an application that assists the user in prioritizing their schedules. The assignments themselves will be displayed in either red, yellow, green, or black depending on their chosen priority. This gives a color coordinated look and makes it simpler for the user to see which assignments are higher priorities.

EDIT/DELETE/VIEW FEATURE

The show assignments page has two features next to each assignment listed that allows the user to either edit or delete each assignment. Each assignment has a corresponding "delete,""edit," and "view" button, which is associated with the assignment's ID (Indicated in Image 3). When the user clicks on the delete button next to a particular assignment, it deletes the assignment from the database by searching for the associated ID.

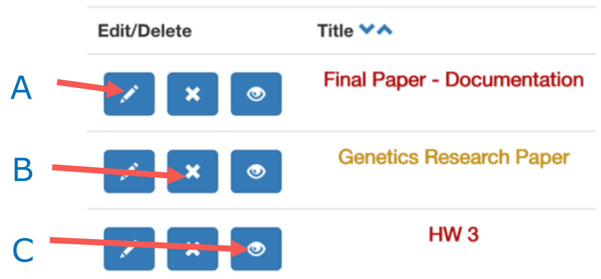


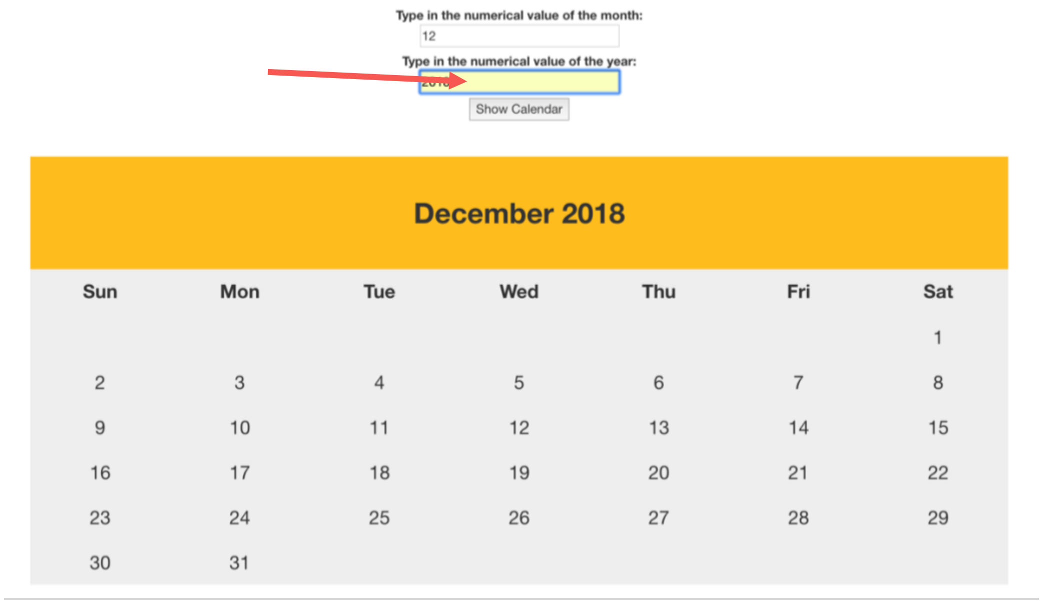
Image 4. Edit, View, and Delete features appears next to each individual assignment. A) Edit option, B) Delete option, and C) View option.

The “edit” button, once clicked, redirects the user to another page, `EditAssignment.html`, where the user can then edit the assignment via a form. This form contains values for the inputs which the user has previously entered. Once the user changes the desired features, they can then submit the form, which redirects them back to the show assignment page along with posting the new changes.

The “view” button function similarly redirects the user to another page, `full_view.html`, where the user can then view the full description of one assignment. This feature is useful for users that have longer assignment descriptions which exceed the maximum display of 25 characters on the show assignments page.

CALENDAR FEATURE

By clicking on the icon in the side navbar that looks like a calendar, it will take you to the main calendar page (Image 1). Here, there will be a form asking for the numerical value of a month and a year. Once the user enters in a value, the program will return a calendar according to that input month and year (Image 5). Below the calendar, there will be a list of assignments, similar to the “show assignments” page. However, the list will display all of the assignments they have based on their due date for that month. Assignments that do not match the month chosen will not be shown here.


Type in the numerical value of the month:

12

Type in the numerical value of the year:

2018

December 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

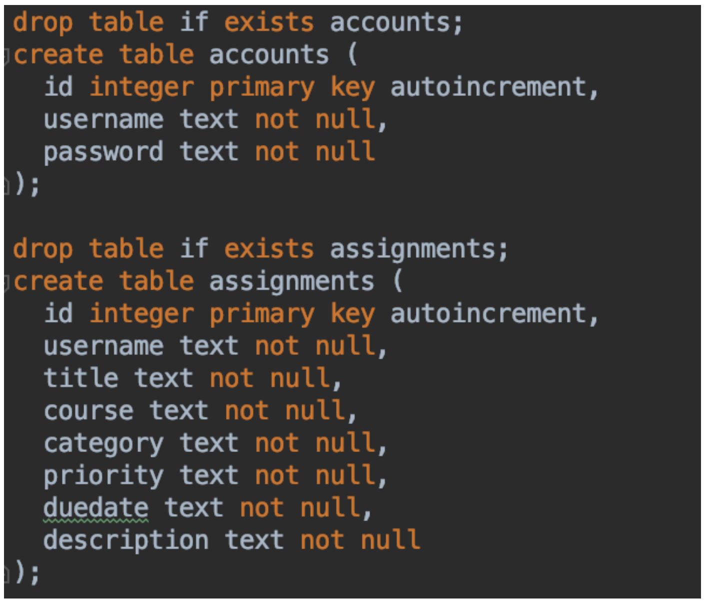
Image 5. Calendar display of December, 2018. The arrow indicates the field that allows the user to adjust the calendar layout.

CODE DOCUMENTATION

Much of the functionality of the application is based in Python and can be found in `app.py`. Several dependencies of this application include Jinja, Flask, Bootstrap, GitHub, and Travis CI. Jinja is a template framework for Python, while Flask is a microframework for Python, which was used frequently throughout the application. Much of the code is based off of an application called "flaskr," which can be further analyzed at the Flask Python Microframework website. Bootstrap was used for many of the styling features of the application and Travis CI was synced to the GitHub project and automatically ran test cases when code gets pushed to GitHub.

DATABASE SCHEMA

The database schema, saved as `schema.sql`, consists of data for two separate tables: `accounts` and `assignments`. Image 6 below displays the content of the `schema.sql` file. The `accounts` table stores a user's id, username, and password. This is especially important for creation of sessions, as it has all of the different user accounts saved.



```sql
drop table if exists accounts;
create table accounts (
  id integer primary key autoincrement,
  username text not null,
  password text not null
);

drop table if exists assignments;
create table assignments (
  id integer primary key autoincrement,
  username text not null,
  title text not null,
  course text not null,
  category text not null,
  priority text not null,
  duedate text not null,
  description text not null
);
```

Image 6. SQL Schema consisting of two tables (Accounts and Assignments) in `schema.sql`.

The other table, `assignments`, stores an assignment id, title, course, category, priority, duedate, description, and the username of the individual who created it. In order to display the correct assignments for each user, having a username variable saved for each assignment gives a parameter to select from the database.

DATABASE INITIATION AND TERMINATION

The database is a crucial part of the application, as it stores important information pertaining to the users and their content. The following commands are found within `app.py` and serve to initialize and terminate the database.

- **`connect_db()`**: Command to connect the database to the application before initializing.
- **`init_db()`**: Initializes the database. This command should be run once at the beginning of deploying the application. Otherwise, the database will be erased.

- **`initdb_command()`**: The command that calls the initializing function. This also returns a print statement acknowledging the initialization of the database.
- **`get_db()`**: Essentially creates a new connection to the database if one was not established previously.
- **`close_db()`**: Terminates the database.

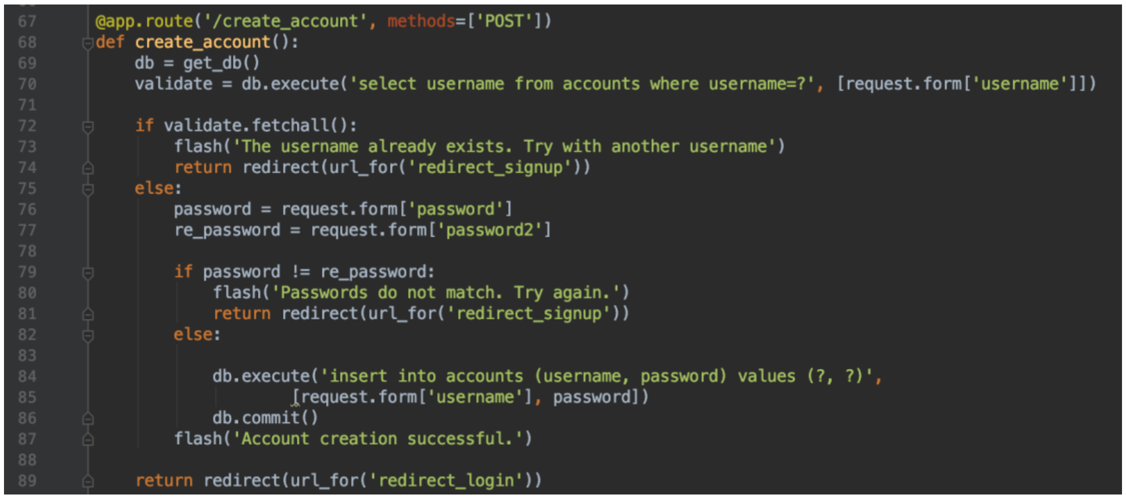
The above commands are directly based off of the flaskr application as previously mentioned. For more information, refer to Flaskr Application Setup Code.

SESSIONS AND AUTHORIZATION

Authorization is a major component of the application, as it allows each individual user to add and change their own assignments. The three major components of authorization are comprised of creating user accounts, logging in, and logging out.

First, the user is required to make an account. The `CreateAccount.html` file consists of a form in which an individual is prompted to enter in a username and password in the appropriate fields (see "Session Feature" above). The function `create_account()` first gets the information stored in the database and saves any account as a variable called "validate" where the username in the database matches the username entered in the input field (Image 7, lines 67-70). The code then checks to see if the username entered matches any entries. If so, then an error appears and the user is redirected to the sign-in page (Image 7, lines 72-74).

If the username is unique, then the function will store the first and second password entries as two separate variables. There is another conditional statement that checks to see if the two passwords entered match one another (Image 7, lines 75-77). This is an additional precautionary for the user in case they mistyped their password. In the case that they do not match, the user will get an error message and be redirected to the sign-in page again (Image 7, lines 79-81). However, if the two passwords do match, then the user's account information will then be inserted into the database and committed. A flash message will appear, saying that the action was successful and the user will then be redirected to the login page (Image 7, lines 82-89).



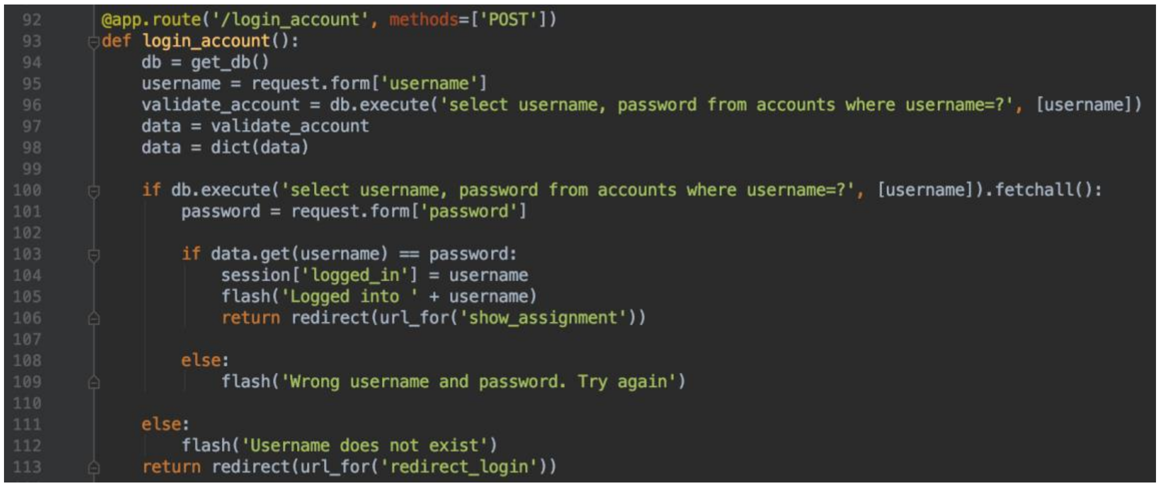
```
67  @app.route('/create_account', methods=['POST'])
68  def create_account():
69      db = get_db()
70      validate = db.execute('select username from accounts where username=?', [request.form['username']])
71
72      if validate.fetchall():
73          flash('The username already exists. Try with another username')
74          return redirect(url_for('redirect_signup'))
75      else:
76          password = request.form['password']
77          re_password = request.form['password2']
78
79          if password != re_password:
80              flash('Passwords do not match. Try again.')
81              return redirect(url_for('redirect_signup'))
82          else:
83
84              db.execute('insert into accounts (username, password) values (?, ?)',
85                      [request.form['username'], password])
86              db.commit()
87          flash('Account creation successful.')
88
89      return redirect(url_for('redirect_login'))
```

Image 7. Create Account feature in `app.py`.

Once the user has completed creating their account and are redirected to the login page, they are then prompted with another form that asks for the username and password they just created. The `login_account()` function first gets the database information and stores the inputted username as a variable. Another variable is saved as accounts which are selected from the database based on their similarity to the username that was just entered by the user. If an account does not exist matching the username entered, then there will be an error and the user will be redirected to the login page (Image 8, lines 111-113). If there exists an account that matches the database, then the function continues to the next conditional statement to check for the correct password.

The conditional statement checks for the same password as the selected username account. If the password matches the one saved in the database, then the username is then attributed to the current user session and the user will be redirected to their homepage displaying a flash message stating the login was successful (Image 8, lines 103-

106). However, if the password does not match the one saved in the user's account, then an error message will appear and they will be redirected to the login once again (Image 8, lines 108-109, 113).



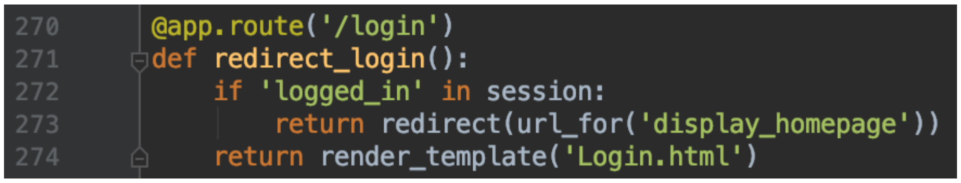
```
92   @app.route('/login_account', methods=['POST'])
93   def login_account():
94       db = get_db()
95       username = request.form['username']
96       validate_account = db.execute('select username, password from accounts where username=?', [username])
97       data = validate_account
98       data = dict(data)
99
100      if db.execute('select username, password from accounts where username=?', [username]).fetchall():
101          password = request.form['password']
102
103          if data.get(username) == password:
104              session['logged_in'] = username
105              flash('Logged into ' + username)
106              return redirect(url_for('show_assignment'))
107
108          else:
109              flash('Wrong username and password. Try again')
110
111      else:
112          flash('Username does not exist')
113      return redirect(url_for('redirect_login'))
```

Image 8. Login feature in `app.py`.

The logout feature is based off of the Flaskr Logout demonstration. The majority of this component of the software is also heavily based off of the Flaskr app, as previously mentioned. For more information, see Flaskr Login and Logout.

REDIRECTING

In some cases, we do not want the user to access particular pages without the proper authorization. In these cases, we added conditional statements to some of the application features. These conditional statements redirect the user to another page if they do not have the proper authorization. An example is shown in image 9, where if the user searches for the login page via the route '`/login`' while the user is already logged in, then this code redirects the user back to their homepage. If the user is not logged in, then they will be redirected to the login page via `Login.html` (Image 9).



```python
@app.route('/login')
def redirect_login():
    if 'logged_in' in session:
        return redirect(url_for('display_homepage'))
    return render_template('Login.html')
```

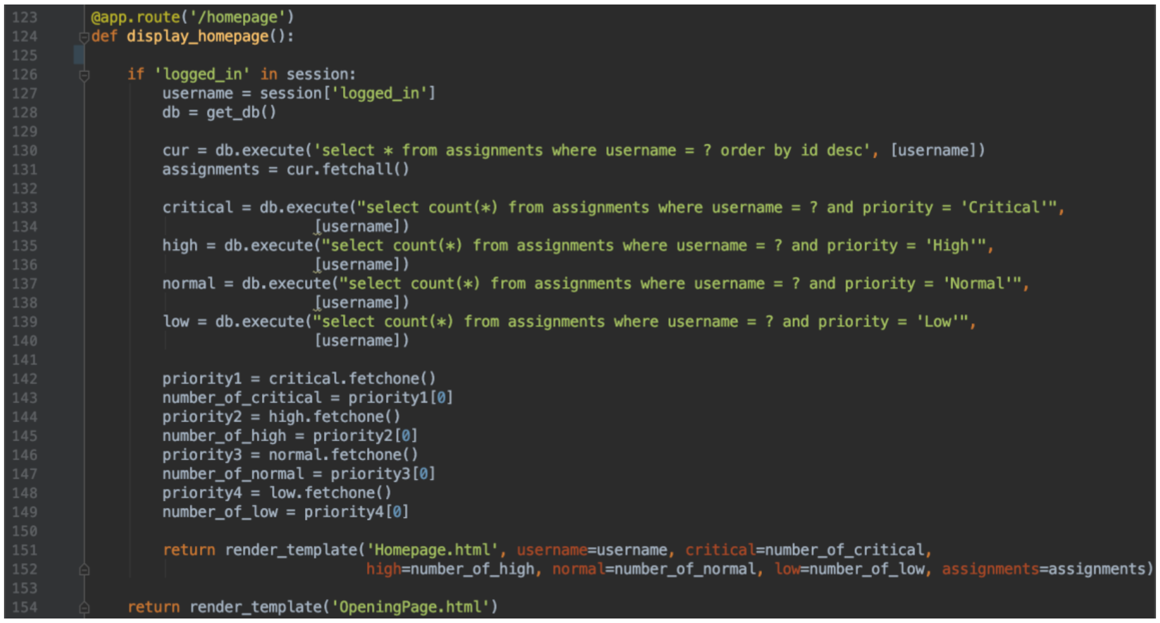
Image 9. Python code for redirecting to login page with a conditional statement, checking for user session in `app.py`.

The `redirect_login()` function is not the only redirecting code, however. There are similar functions for the '`/add`', '`/signup`', and '`/`' routes as well. This is to prevent users from either logging in or creating an account when they are already logged in, or to prevent users from reaching pages in which they do not have the authorization to access, such as the add assignment page.

HOMEPAGE FUNCTION

When a user initially logs in, they are directed to the homepage (`Homepage.html`). This page displays a "Welcome" message with the user's username, along with a brief list of some assignments and a numerical value for assignments with each priority attribute (see "Dashboard Feature" above). The `Homepage.html` code is mainly divided into two columns, which are the `right-col` and the `left-col`. The homework display is in the right column, while the priority display is in the left. Prior to rendering the template, the `display_homepage()` function must be called.

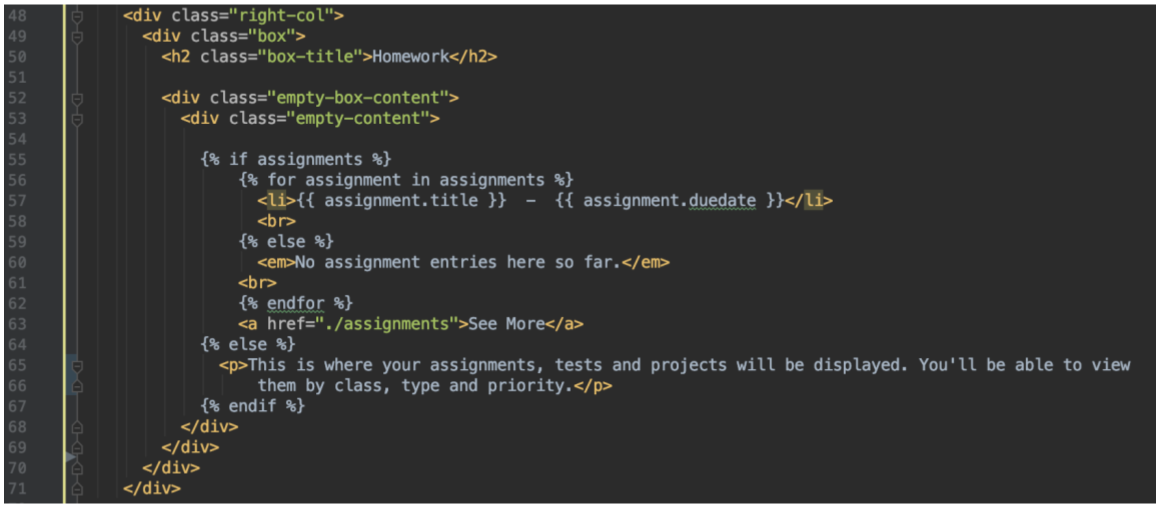
The function to display the homepage first requires that the user is logged in (Image 10, line 126). If the user isn't logged in, then they will be redirected to the opening page. However, if the user is logged in, then first the username will be saved and the database information will be retrieved (Image 10, lines 127-128). Then, all assignments that contain the same value for the current user's username will be saved as a variable (Image 10, line 130-131). For each of the priority values, a similar command is made where all of the assignments that are also associated with the user and match the particular chosen priority will be saved in separate variables (Image 10, line 133-140). In this case, however, the number of assignments matching that criteria will be saved in the variable, as opposed to the assignments as a whole (Image 10, line 142-149).



```
123 @app.route('/homepage')
124 def display_homepage():
125 
126     if 'logged_in' in session:
127         username = session['logged_in']
128         db = get_db()
129 
130         cur = db.execute('select * from assignments where username = ? order by id desc', [username])
131         assignments = cur.fetchall()
132 
133         critical = db.execute("select count(*) from assignments where username = ? and priority = 'Critical'",
134                              [username])
135         high = db.execute("select count(*) from assignments where username = ? and priority = 'High'",
136                              [username])
137         normal = db.execute("select count(*) from assignments where username = ? and priority = 'Normal'",
138                              [username])
139         low = db.execute("select count(*) from assignments where username = ? and priority = 'Low'",
140                              [username])
141 
142         priority1 = critical.fetchone()
143         number_of_critical = priority1[0]
144         priority2 = high.fetchone()
145         number_of_high = priority2[0]
146         priority3 = normal.fetchone()
147         number_of_normal = priority3[0]
148         priority4 = low.fetchone()
149         number_of_low = priority4[0]
150 
151         return render_template('Homepage.html', username=username, critical=number_of_critical,
152                                high=number_of_high, normal=number_of_normal, low=number_of_low, assignments=assignments)
153 
154     return render_template('OpeningPage.html')
```

Image 10. Function for homepage (display_homepage() function) in `app.py`.

The values for the priorities and assignments are then saved as variables and are used as values within the html template when rendered (Image 10, lines 151-152). These values can then be utilized when displaying the user's assignments or priorities. The homepage column containing the list of assignments first has a conditional statement where if some assignments exist, then it will continue to the next conditional statement. However, if there are no assignments, then the software will return the paragraph in lines 65-66 from image 11, which displays a simple informative message about the column. If there are assignments though, then the software will loop through the list of assignments and add the title of each assignment and the corresponding duedate separated by a dash (Image 11, lines 56-58). Once the for loop ends, there is a link to the show assignments page, where the rest of the assignment information can be viewed (Image 11, line 63).



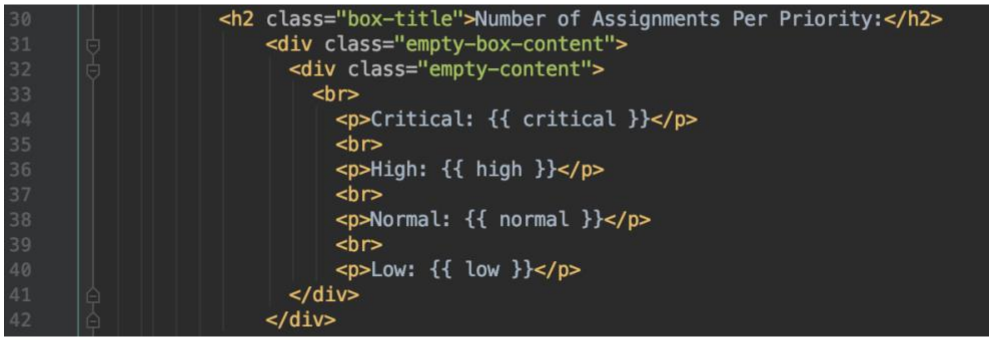
```
<div class="right-col">
  <div class="box">
    <h2 class="box-title">Homework</h2>

    <div class="empty-box-content">
      <div class="empty-content">

        {% if assignments %}
            {% for assignment in assignments %}
              <li>{{ assignment.title }}  –  {{ assignment.duedate }}</li>
              <br>
            {% else %}
              <em>No assignment entries here so far.</em>
            <br>
            {% endfor %}
            <a href="./assignments">See More</a>
        {% else %}
          <p>This is where your assignments, tests and projects will be displayed. You'll be able to view
              them by class, type and priority.</p>
        {% endif %}
      </div>
    </div>
  </div>
</div>
```

Image 11. Column of assignments in `Homepage.html`.

Unlike the previous case for assignments, there are no conditional statements to display the priority number, as seen in image 12. Instead, the value for each condition is called from the python function and placed in the template, as observed in lines 34, 36, 38, and 40. If there are no assignments that match any of the categories, then each priority will simply be displayed as “0.”

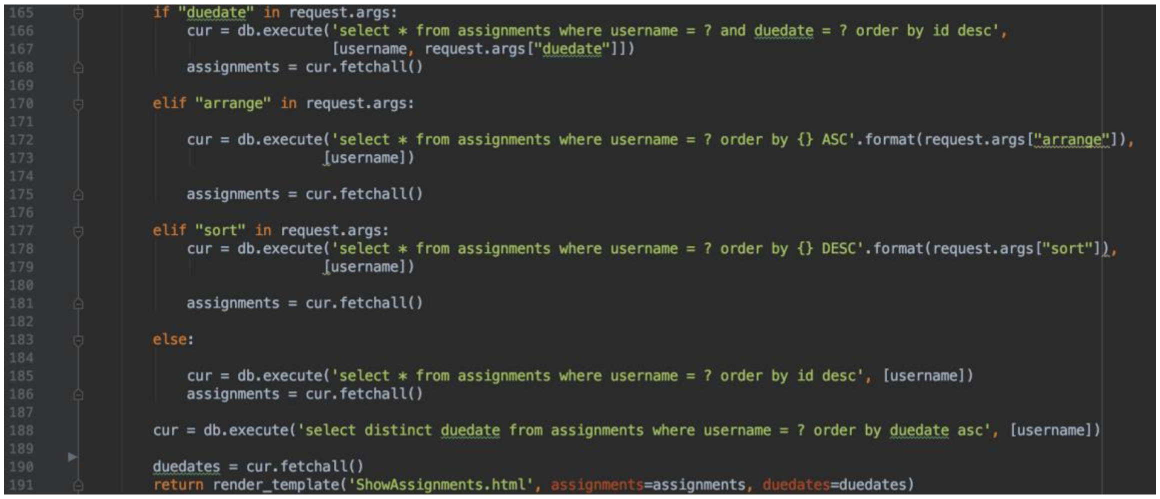


```
30  <h2 class="box-title">Number of Assignments Per Priority:</h2>
31    <div class="empty-box-content">
32     <div class="empty-content">
33      <br>
34       <p>Critical: {{ critical }}</p>
35       <br>
36       <p>High: {{ high }}</p>
37       <br>
38       <p>Normal: {{ normal }}</p>
39       <br>
40       <p>Low: {{ low }}</p>
41     </div>
42    </div>
```

Image 12. Column of assignment priorities in `Homepage.html`.

SHOW ASSIGNMENTS FUNCTION

The show assignments page has several features. These include displaying the assignments, sorting the assignments based on different parameters, editing assignment, deleting assignments, and viewing individual assignments. First, if the user is logged in, then the function will get the database information and sort it by several conditions. The initial function when a user gets to the show assignment page is to get all assignments that match the user's username (Image 13, lines 185-186). However, when a user is on the page, they have several options.

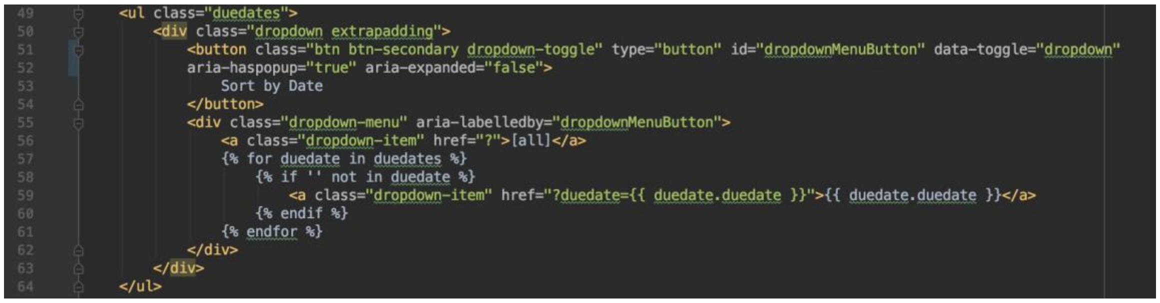


```
165     if "duedate" in request.args:
166         cur = db.execute('select * from assignments where username = ? and duedate = ? order by id desc',
167                          [username, request.args["duedate"]])
168         assignments = cur.fetchall()
169
170     elif "arrange" in request.args:
171
172         cur = db.execute('select * from assignments where username = ? order by {} ASC'.format(request.args["arrange"]),
173                         [username])
174
175         assignments = cur.fetchall()
176
177     elif "sort" in request.args:
178         cur = db.execute('select * from assignments where username = ? order by {} DESC'.format(request.args["sort"]),
179                         [username])
180
181         assignments = cur.fetchall()
182
183     else:
184
185         cur = db.execute('select * from assignments where username = ? order by id desc', [username])
186         assignments = cur.fetchall()
187
188     cur = db.execute('select distinct duedate from assignments where username = ? order by duedate asc', [username])
189
190     duedates = cur.fetchall()
191     return render_template('ShowAssignments.html', assignments=assignments, duedates=duedates)
```

Image 13. Function for show assignments page (`show_assignment()` function) in `app.py`.

The first of these options on the show assignments page is to display all of the assignments based on individual due dates. In order to do this, a dropdown button was added to the `ShowAssignments.html` file, displaying all due dates from the assignments matching the user's username (see Image 14 below). As a side note, both lines 186 and 190 in image 13 save nearly the same assignment list as different variables. However, the `duedates` variable (line 190) only saves one unique assignment date per each assignment. This is to prevent multiple copies of the same due date from being displayed.

Going back to the due date dropdown button, a list of all due dates associated with at least one assignment is enclosed within a link tag. When the user clicks on one of the duedates, it redirects them and calls the `show_assignment()` function again. However, the conditional statement in lines 165-168 of Image 13 is then met. This then saves all assignments that match the due date chosen into the `assignments` variable. When the show assignments page is then loaded, it will display all assignments with the chosen due date.



```
<ul class="duedates">
    <div class="dropdown extrapadding">
        <button class="btn btn-secondary dropdown-toggle" type="button" id="dropdownMenuButton" data-toggle="dropdown"
        aria-haspopup="true" aria-expanded="false">
            Sort by Date
        </button>
        <div class="dropdown-menu" aria-labelledby="dropdownMenuButton">
            <a class="dropdown-item" href="?">[all]</a>
            {% for duedate in duedates %}
                {% if '' not in duedate %}
                    <a class="dropdown-item" href="?duedate={{ duedate.duedate }}">{{ duedate.duedate }}</a>
                {% endif %}
            {% endfor %}
        </div>
    </div>
</ul>
```

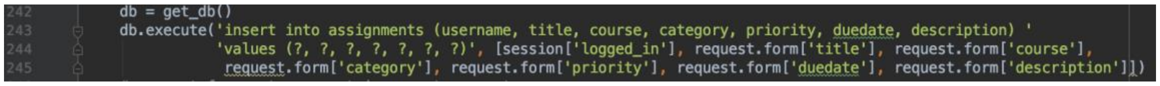
Image 14. “Sort by Date” dropdown button in `ShowAssignments.html`

Similarly, upon clicking an arrow next to one of the assignment columns (see “Show Assignments Feature” above), the sort function for each column redirects the user to the `show_assignment()` function and either executes the `sort` or `arrange` condition (depending on which arrow they clicked). One function sorts the assignments in ascending order based on the chosen parameter, and the other function sorts the assignments in descending order (Image 13, lines 170-181). Then the ordered list is returned via the `assignments` variable and rendered in the show assignments page.

ADD ASSIGNMENTS FUNCTION

The add assignments feature is similar to the edit feature. When a user clicks on a button linking to the add assignments page, the `AddAssignments.html` file is then rendered. Here, the user is able to input several assignment variables. Once the user submits their information via a submit button, then the `add_assignment()` function inserts each of those variables into the database as a new entry (Image

15, lines 242-245). Then, the user is redirected to the show assignments page, where their new assignment will be displayed.

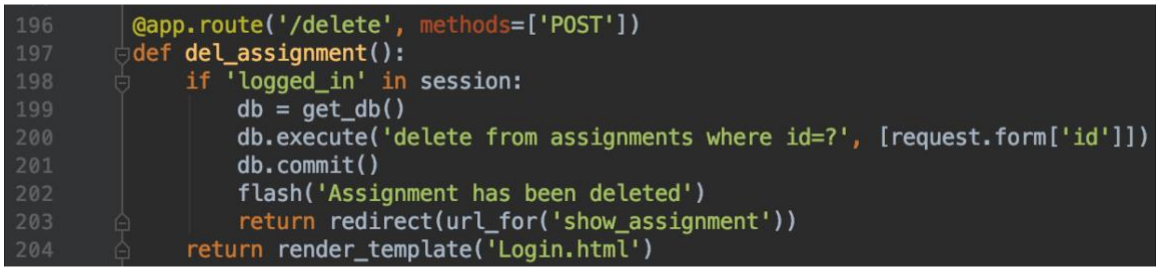


```
242         db = get_db()
243         db.execute('insert into assignments (username, title, course, category, priority, duedate, description) '
244                    'values (?, ?, ?, ?, ?, ?, ?)', [session['logged_in'], request.form['title'], request.form['course'],
245                     request.form['category'], request.form['priority'], request.form['duedate'], request.form['description']])
```

Image 15. Code for `add_assignment()` function in `app.py` displaying the insert SQL statement.

EDIT/DELETE FUNCTION

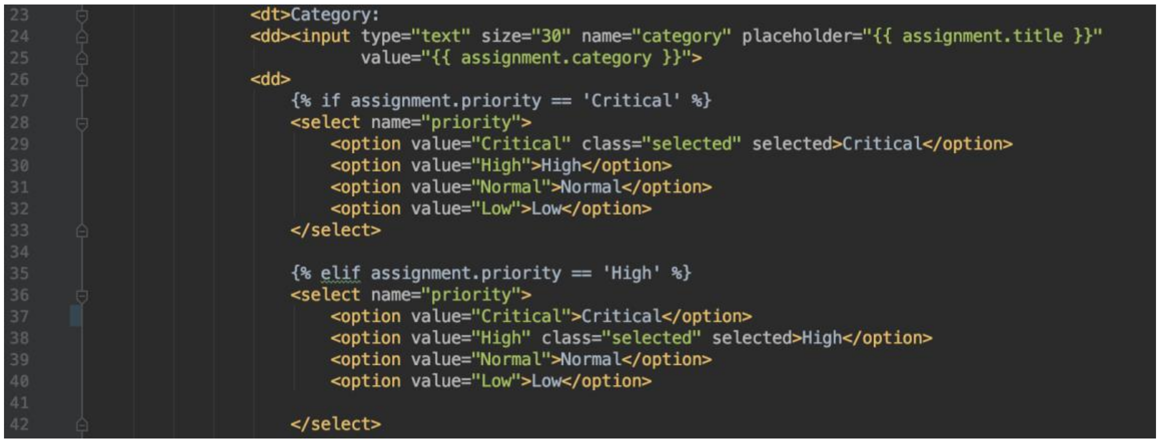
As previously mentioned, the edit and delete functions are accessible via the show assignments page and the calendar page. When a user selects the delete button associated with an individual assignment, a post request is made (Image 16, line 196). The function will then delete the assignment matching the id of the assignment that the user selected via the SQL statement in line 200 of image 16.



```
196 @app.route('/delete', methods=['POST'])
197 def del_assignment():
198     if 'logged_in' in session:
199         db = get_db()
200         db.execute('delete from assignments where id=?', [request.form['id']])
201         db.commit()
202         flash('Assignment has been deleted')
203         return redirect(url_for('show_assignment'))
204     return render_template('Login.html')
```

Image 16. Code for `del_assignment()` function in `app.py`.

The edit function is much more similar to the add assignments feature in that it redirects the user to a new file with a form that performs an SQL request. When a user selects the edit button on one of the assignments, they are then redirected to the `EditAssignment.html` file via the `edit_entry()` function (refer to `app.py` file). However, the form inputs already have some of the user's assignment values already displayed. This way, a user can see what is already saved for that assignment and it also helps the user save time when making minor changes. Lines 23-25 of image 17 provides an example of what an input field looks like with a placeholder value.



```
23          <dt>Category:
24          <dd><input type="text" size="30" name="category" placeholder="{{ assignment.title }}"
25                     value="{{ assignment.category }}">
26          <dd>
27              {% if assignment.priority == 'Critical' %}
28              <select name="priority">
29                  <option value="Critical" class="selected" selected>Critical</option>
30                  <option value="High">High</option>
31                  <option value="Normal">Normal</option>
32                  <option value="Low">Low</option>
33              </select>
34
35              {% elif assignment.priority == 'High' %}
36              <select name="priority">
37                  <option value="Critical">Critical</option>
38                  <option value="High" class="selected" selected>High</option>
39                  <option value="Normal">Normal</option>
40                  <option value="Low">Low</option>
41
42              </select>
```

Image 17. `EditAssignment.html` code showing priority condition and example of entry placeholder.

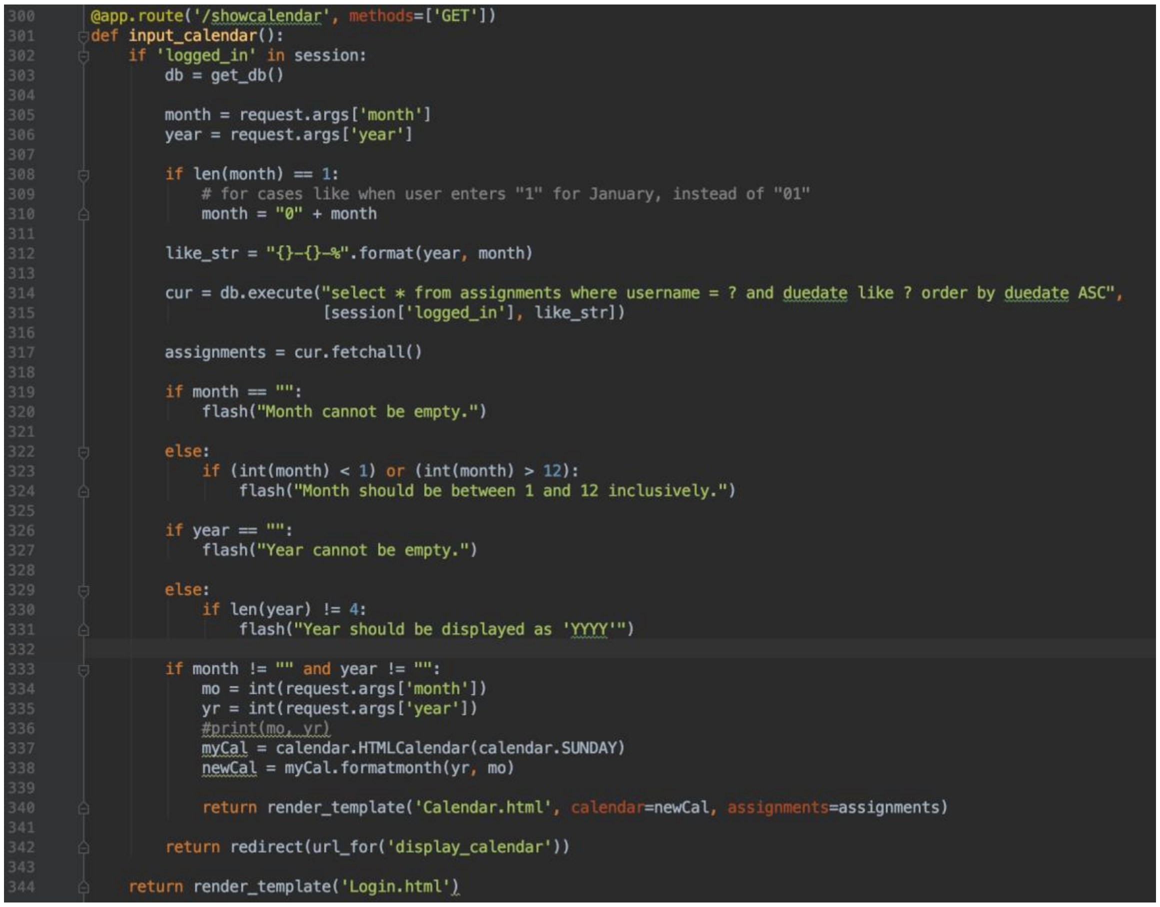
The priority feature requires a conditional statement in order to show to user which priority they currently have selected for that particular assignment. The code in lines 27-42 of image 17 above shows two of the four conditional statements. This condition checks to see the value saved as the priority variable in that assignment. If it matches the first statement in line 27, then it will display the "Critical" priority as selected. If the variable matches one of the other priorities, it will display that priority (example in lines 35-42 of image 17).

Once the user submits their changes via a submit button in the form, then an SQL "update" statement with each parameter will be called. The updates will be committed to the database, and the user will be redirected back to the show assignments page with their updates being displayed.

CALENDAR FUNCTION

The calendar feature can be accessed via the `sidenav` bar, as explained in the Calendar Feature section above. Once the `Calendar.html` template has been requested via the `display_calendar()` function in `app.py`, the user will be faced with a form requiring the month and year. Once the form is submitted, a function called `input_calendar()` is then called (Image 18). This function takes in the month and year inputs and saves them as variables (Image 18, lines 305-306). If a month was entered as a single digit instead of the "MM" format, then a "0" will be added in front of the entered month value and saved.

The month and year are then formatted as inputs that can be entered in an SQL "like" statement, where it searches the database for matching entries starting with the year and month entered (the "%" in line 312 is necessary to indicate that the following information does not matter, as long as the preceding code matches). This new format is saved as a string variable and inserted into an SQL statement with a "like" clause (Image 18, lines 314-315). The assignment list is then used in the `Calendar.html` template to display all of the assignments using similar code to that of `ShowAssignments.html`.



```
300  @app.route('/showcalendar', methods=['GET'])
301  def input_calendar():
302      if 'logged_in' in session:
303          db = get_db()
304
305          month = request.args['month']
306          year = request.args['year']
307
308          if len(month) == 1:
309              # for cases like when user enters "1" for January, instead of "01"
310              month = "0" + month
311
312          like_str = "{}-{}-%".format(year, month)
313
314          cur = db.execute("select * from assignments where username = ? and duedate like ? order by duedate ASC",
315                           [session['logged_in'], like_str])
316
317          assignments = cur.fetchall()
318
319          if month == "":
320              flash("Month cannot be empty.")
321
322          else:
323              if (int(month) < 1) or (int(month) > 12):
324                  flash("Month should be between 1 and 12 inclusively.")
325
326          if year == "":
327              flash("Year cannot be empty.")
328
329          else:
330              if len(year) != 4:
331                  flash("Year should be displayed as 'YYYY'")
332
333          if month != "" and year != "":
334              mo = int(request.args['month'])
335              yr = int(request.args['year'])
336              #print(mo, yr)
337              myCal = calendar.HTMLCalendar(calendar.SUNDAY)
338              newCal = myCal.formatmonth(yr, mo)
339
340              return render_template('Calendar.html', calendar=newCal, assignments=assignments)
341
342          return redirect(url_for('display_calendar'))
343
344      return render_template('Login.html')
```

Image 18. Code for `input_calendar()` function in `app.py`.

While this function fetches the assignments that match the date entered, the calendar itself still needs to be generated. The first condition in line 319 of image 18 checks to see if the month is empty and displays and error message if so. The second condition (lines 322-324) checks to see if the month is within the correct restraints (1-12) and also displays an error message. The third conditional statement (lines 326-327) checks to see if the year is empty and, likewise, displays an error message if it is true. The fourth conditional statement (lines 329-331) checks to see if the year is four digits long. While the code will still run if the year is more or less than four digits, it is impractical to allow those parameters, as years other than four digits will not make sense in the timespan of our application.

Finally, if the month and the year both have a valid input, then they will be saved as integer values and used as inputs in the python calendar feature. This calendar function is a module from a python library known as `calendar`, which must be imported first. Line 337 generates the HTML for the calendar, displaying Sunday as the start of the week. Line 338 then inputs the correct dates according to the month and year selected. This is then saved as a variable and returned in the `Calendar.html` file.