

# Creating a To Do List with Bottle

## Contents

Introduction.....	2
Setting up .....	2
Create the Base App .....	3
Add a Home Page Template.....	4
Create To Do List Table.....	5
Insert Some Data.....	8
Display Current To Do List .....	10
Add New To Do Item .....	12
Mark Item as Completed.....	15
Remove To Do Item .....	17
Adding Some Styling.....	19
Further Discussion.....	20
Appendix: Code Listing .....	21
File Structure .....	21
todo_app.py .....	21
index.tpl .....	24
style.css.....	26

## Introduction

In this tutorial, we are going to create a really simple To Do app using the Python Bottle framework. This will demonstrate how to:

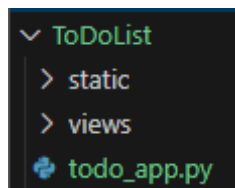
- Create a simple application and run it on your local computer
- Set up a home page using a template
- Create a new, empty database
- Insert some test data into the database
- Retrieve the records from a table and display them on the webpage
- Insert a new record into a database
- Delete a record from a database

This tutorial assumes that you have completed the **Getting Started with Bottle** tutorial and are familiar with the basics of how a Bottle app works.

## Setting up

1. Open your **UsingBottle** folder in VS Code
2. Inside your **UsingBottle** folder, create a new folder called **ToDoList**
3. Inside your **ToDoList** folder, create:
  - a. A new file called **todo\_app.py**
  - b. A new folder called **views**
  - c. A new folder called **static**

You should have a folder structure something like this:



## Create the Base App

To get started, we first need to set up the basic web app

1. Open the **todo\_app.py** file and add the following code to set up the basic Bottle app:

```
from bottle import route, run

#####
#
# Routes for each page
#
# Home page
@route('/')
def index():
    return '<h1>My To Do List</h1>'

#####
#
# Helper functions

#####
#
# Run the web server to serve up the pages
#
run(host='localhost', port=8081, debug=True, reloader=True)
```

2. Go to the terminal in VS Code and change the current working directory to the **ToDoList** directory.

NOTE: The easiest way to do this is:

- right click on the ToDoList folder in the VS Code explorer and select **Copy Path**
- go to the terminal and type **cd <path>** where <path> is the path that you just copied (paste the path)
- press **Enter** to run the cd command

3. Run your web app using the command:

**python todo\_app.py**

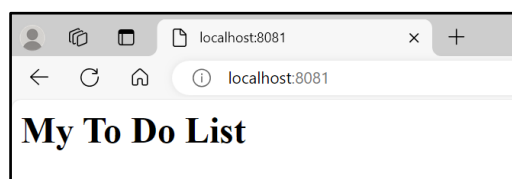
4. Open your web browser and navigate to the page:

**localhost:8081**

NOTE: If this does not work, try navigating to **127.0.0.1:8081**

NOTE: You can change the port you are using for the web app on the last line of the **todo\_app.py** file.

You should now see something like this in your browser window:



## Add a Home Page Template

We are now going to add a home page for our To Do list application using a template.

1. Inside the **views** folder, create a new file and name it **index.tpl**
2. Open the **index.tpl** file and add the following HTML:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My To Do List</title>
  </head>
  <body>
    <h1>My To Do List</h1>
    <footer>A To Do list using the Bottle framework</footer>
  </body>
</html>
```

3. Go back to the **todo\_app.py** file and:
  - a. Update the **import** line to include the **template** function.
  - b. Modify the **index** function that that it returns the **index** template.
4. Your **todo\_app.py** file should now look something like this:

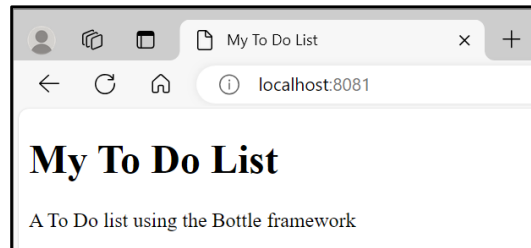
```
from bottle import route, run, template

#####
#
# Routes for each page
#
# Home page
@route('/')
def index():
    return template('index')

#####
#
# Helper functions

#####
#
# Run the web server to serve up the pages
#
run(host='localhost', port=8081, debug=True, reloader=True)
```

5. Make sure that the web app is running and reload the browser page. Your page should now look something like this:



## Create To Do List Table

Now that we have the basic app up and running, we can start building the database to store our to do list. This is going to be a very simple database, with a single table to store each item on our to do list.

The data dictionary for the table we are going to use is given below:

Element Name	Data Type	Size	Description	Constraint
task_id	Integer		Unique identifier for each task	Required. Automatically created when new task added.
title	Text	30	Title for task	Required
description	Text	300	Description of task	Optional
completed	Boolean		Indicate if task has been completed	Default value is FALSE

The steps required to create the database are:

- Import sqlite3 to the main app file
- Add a constant to the
- Add a new route and function to the main app file
- Inside the function:
  - Create a connection to the database file
  - Drop the existing table (if there is one)
  - Create a new table
  - Commit the changes and close the connection
- Add a link on the homepage to run the function that was just written

Follow these steps:

1. Go to the **todo\_app.py** file
2. Add an **import** statement at the start of the file to import `sqlite3`

```
import sqlite3
```

3. In the routes section, use the code below to add a new route below the index route

```
# Create new To Do list (remove existing list and create new empty table)
@route('/create_todo_list')
def create_todo_list():
    connection = sqlite3.connect('todo_list.sqlite')
    cursor = connection.cursor()

    query = 'DROP TABLE IF EXISTS Task'
    cursor.execute(query)

    query = '''
        CREATE TABLE Task
            (task_id INTEGER PRIMARY KEY,
             title TEXT NOT NULL,
             description TEXT,
             completed BOOLEAN DEFAULT FALSE)
    '''
    cursor.execute(query)

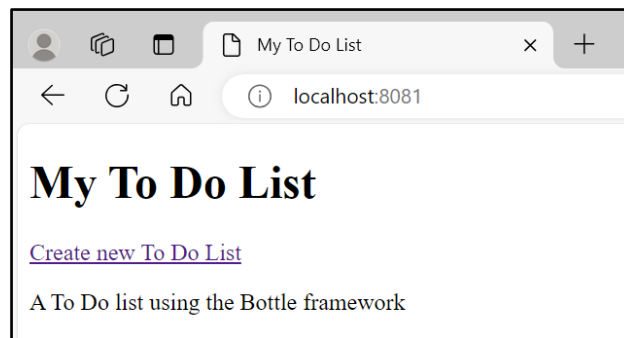
    connection.commit()
    connection.close()

    return template('index')
```

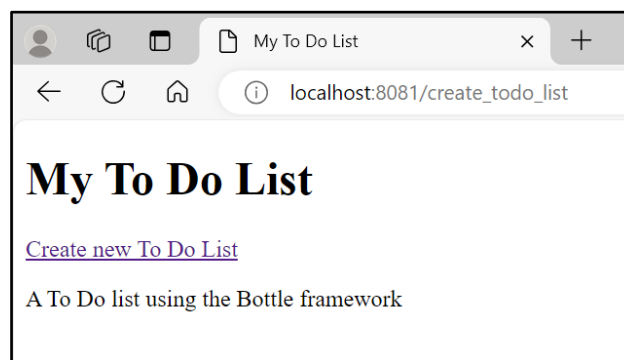
4. Now that we have added the route to the app, we need to add a link to the index file to call the new function. Go to the **index.tpl** file and use the following HTML between the heading (the `h1` tags) and the footer.

```
<section>
    <p><a href="/create_todo_list">Create new To Do List</a></p>
</section>
```

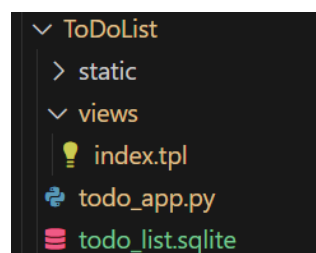
- Once again, make sure the web app is running and reload the page. Your page should look something like this:



- Click on the link to run the new script and create the new database. The page will not seem to change, but have a look at the address bar and you should notice that the URL has changed to include **/create\_todo\_list** as in the image below.



You should also notice that there has been a new file created in the **ToDoList** folder called **todo\_list.sqlite**, as in the image below.



## Insert Some Data

Once the database file has been created with a table to store each task, we can insert some sample tasks. To do this we will follow a similar process to the previous step of creating the table:

- Add a route and function to the main app
- Inside the function:
  - Create a connection to the database
  - Insert the data
  - Commit the changes and close the connection
- Add a link to the index page to call the route

Follow these steps:

1. Go to the **todo\_app.py** file
2. In the routes section, use the code below to add a new route below the *create\_todo\_list* route

```
@route('/insert_sample_data')
def insert_sample_data():
    connection = sqlite3.connect('todo_list.sqlite')
    cursor = connection.cursor()

    task1 = { 'title' : 'Wash car',
              'description': 'Take care to car wash and clean the outside',
              'completed' : False }
    task2 = { 'title' : 'Maths',
              'description' : 'Complete exercise 6 on page 231',
              'completed' : True }
    task3 = { 'title' : 'Shopping',
              'description' : 'Buy ingredients to cook spaghetti bolognese
for dinner',
              'completed' : False }

    query = 'INSERT INTO Task (title, description, completed) VALUES (:title,
:description, :completed)'
    cursor.execute(query, task1)
    cursor.execute(query, task2)
    cursor.execute(query, task3)

    connection.commit()
    connection.close()

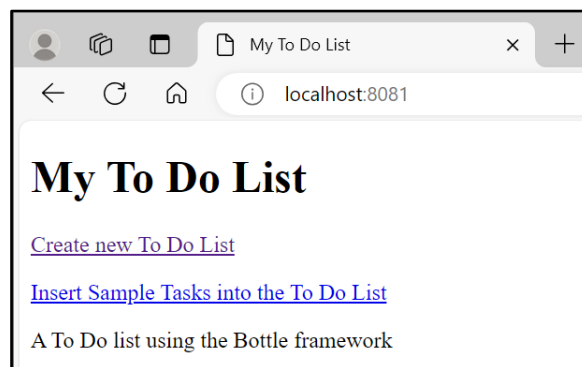
    return template('index')
```



- Now that we have added the route to the app, we need to add a link to the index file to call the new function. Go to the **index.tpl** file and add the following HTML below the section to create the empty list you added previously.

```
<section>
  <p><a href="/insert_sample_data">Insert Sample Tasks into the To
Do List</a></p>
</section>
```

- Once again, make sure the web app is running and reload the page. Your page should look something like this:



- Click on the link to run the new script and create the new database. The page will not seem to change, but have a look at the address bar and you should notice that the URL has changed to include **/insert\_sample\_data** (similar to the previous step).
- Open the database file using your database browser and check to see if the records have been created. There should now be 3 items in the To Do list.  
**NOTE:** Each time you click the link, you will get 3 new items inserted into the list, each with a unique *task\_id*, although the same title and description.

## Display Current To Do List

Now that we have added some items to our To Do list, we need to be able to display the current To Do list on the web page. Once again, we will follow these steps:

- Modify the index page to display the results of a query using an HTML table
- Add a route and function to the main app
- Inside the function:
  - Create a connection to the database
  - Select the data from the Task table
  - Return the index template, passing in the results of the query
  - Close the connection (NOTE: We do not need to *commit* as we have not made any changes to the database)
- Add a link to the index page to call the route

1. Go to the **index.tpl** file
2. Add the following section of code below the section to insert sample data:

```
<section>
  <p><a href="/show_to_do_list">Show the current To Do Items</a></p>
  <h3>My Tasks</h3>
  % if defined('records'):
    % if len(records) < 1:
      <p><strong>You have no items in your To Do list</strong></p>
    % else:
      <table>
        <tr>
          % for field in records[0].keys():
            <th>{{ field }} </th>
          % end
        </tr>
        % for record in records:
          <tr>
            % for field in record:
              <td>{{ field }}</td>
            % end
          </tr>
        % end
      </table>
    % end
  % end
</section>
```

3. Go to the **todo\_app.py** file
4. In the routes section, use the code below to add a new route below the *insert\_sample\_data* route:

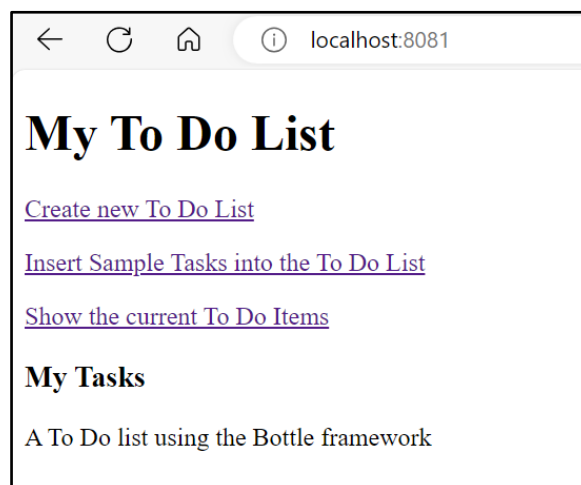
```
# Retrieve all the items in the to do list and display them on the page
@route('/show_to_do_list')
def show_to_do_list():
    connection = sqlite3.connect('todo_list.sqlite')
    connection.row_factory = sqlite3.Row
    cursor = connection.cursor()

    query = 'SELECT * FROM Task'
    cursor.execute(query)
    tasks = cursor.fetchall()

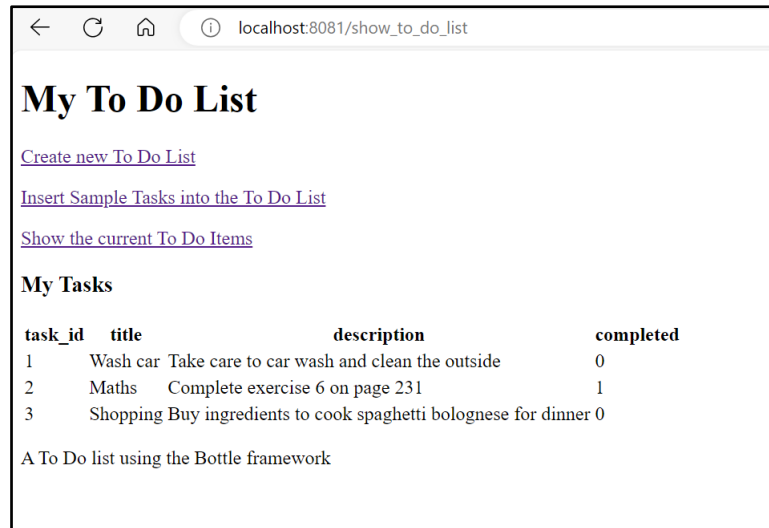
    connection.close()

    return template('index', records=tasks)
```

5. Once again, make sure the web app is running and reload the page. Your page should look something like this:



6. Click on the link to *Show the current To Do Items* and you should get a table of the items that have been inserted into the list, looking something like this:



NOTE: Each time we click on a different link the table of to do items will disappear!

## Add New To Do Item

Now that we have created an empty To Do list, added some sample data and worked out how to display the items on our list, we need to be able to add a new item to our list. To do this we will follow a similar process to inserting the sample data, except this time instead of hard coding the values in, we will get the task name and description from an HTML form. To do this, we will need to follow these basic steps:

- Import the **request** function from Bottle to the main app (as part of the initial **import** statement)
- Add a route and function to the main app. The route should use the HTTP POST method
- Inside the function:
  - Create a connection to the database
  - Get the data from the HTML form that called the route
  - Insert the data
  - Commit the changes and close the connection
- Add an HTML form to the index page to allow the user to enter the correct details and then call the route

1. Go to the **todo\_app.py** file
2. In the routes section, use the code below to add a new route below the *show\_to\_do\_list* route:

```
# Add new task
@route('/add_task', method='POST')
def add_task():
    connection = sqlite3.connect('todo_list.sqlite')
    cursor = connection.cursor()

    # Retrieve the values from the form as a dictionary
    # NOTE: HTML form field names (from the template) must match parameter
names in INSERT query
    form_data = dict(request.params)
    query = 'INSERT INTO Task (title, description) VALUES (:title,
:description)'
    cursor.execute(query, form_data)

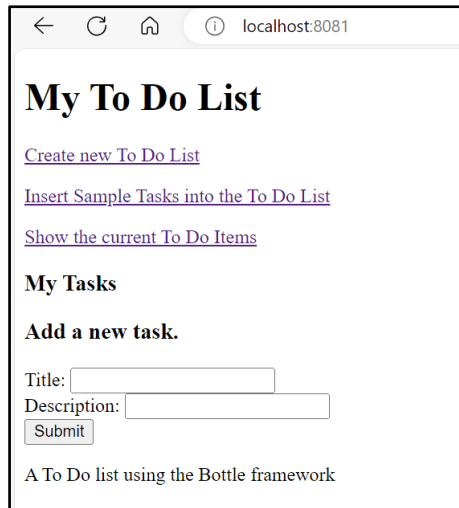
    connection.commit()
    connection.close()

    return template('index')
```

3. Go to the **index.tpl** file
4. Add the following section of code below the section to show the to do list:

```
<section>
    <form action="/add_task" method="post">
        <h3>Add a new task.</h3>
        Title: <input type="text" name="title" /><br />
        Description: <input type="text" name="description" /><br />
        <input type="submit">
    </form>
</section>
```

5. Once again, make sure the web app is running and reload the page. Your page should look something like this:



← ↻ 🏠 ⓘ localhost:8081

# My To Do List

[Create new To Do List](#)

[Insert Sample Tasks into the To Do List](#)

[Show the current To Do Items](#)

## My Tasks

**Add a new task.**

Title:

Description:

A To Do list using the Bottle framework

6. To test your code, add a task and click the **Submit** button. Then click the link to **Show the current To Do Items** and you should see the new task in the list.

## Mark Item as Completed

When we have completed a task on our To Do list, it would be handy to be able to update the list and mark that item as completed. To achieve this, we will follow a similar process to adding a new task, except this time we will use an UPDATE query to modify the task in the database, using a form to tell us which task to update. To do this, we will need to follow these basic steps:

- Add a route and function to the main app. The route should use the HTTP POST method
  - Inside the function:
    - Create a connection to the database
    - Get the task id from the HTML form that called the route
    - Update the correct record in the database
    - Commit the changes and close the connection
  - Add an HTML form to the index page to allow the user to enter the correct details and then call the route
1. Go to the **todo\_app.py** file
  2. In the routes section, use the code below to add a new route below the *add\_task* route:

```
# Mark task as completed
@route('/complete_task', method='POST')
def complete_task():
    connection = sqlite3.connect('todo_list.sqlite')
    cursor = connection.cursor()

    task_id = request.forms.get('task_id')
    values = { 'task_id' : task_id }
    query = '''
        UPDATE Task
        SET completed = TRUE
        WHERE task_id = :task_id
    '''

    cursor.execute(query, values)

    connection.commit()
    connection.close()

    return template('index')
```

3. Go to the **index.tpl** file
4. Add the following section of code below the section to add a new task to the to do list:

```
<section>
    <form action="/complete_task" method="post">
        <h3>Mark a task as completed.</h3>
        Task Id: <input type="text" name="task_id" /><br />
        <input type="submit">
    </form>
</section>
```

5. Once again, make sure the web app is running and reload the page. Your page should look something like this:

← ↻ 🏠 ⓘ localhost:8081

## My To Do List

[Create new To Do List](#)

[Insert Sample Tasks into the To Do List](#)

[Show the current To Do Items](#)

### My Tasks

**Add a new task.**

Title:

Description:

**Mark a task as completed.**

Task Id:

A To Do list using the Bottle framework

- To test your code, add a task id to the form and click the **Submit** button. Then click the link to **Show the current To Do Items** and you should see that the task in your list has been updated and now shows a 1 in the completed column (this is how SQLite represents Boolean values – either 1 or 0).



## Remove To Do Item

Sometimes we may want to tidy up our list and remove old items, or we might make a mistake and do not need an item in our list. In that case, we may want to delete a specific item from the list. To achieve this, we will follow almost exactly the same process as updating the list, but instead of using an UPDATE query, we will use a DELETE query. To do this, we will need to follow these basic steps:

- Add a route and function to the main app. The route should use the HTTP POST method
  - Inside the function:
    - Create a connection to the database
    - Get the task id from the HTML form that called the route
    - Delete the correct record from the database
    - Commit the changes and close the connection
  - Add an HTML form to the index page to allow the user to enter the correct details and then call the route
1. Go to the **todo\_app.py** file
  2. In the routes section, use the code below to add a new route below the *complete\_task* route:

```
# Add new task
@route('/remove_task', method='POST')
def remove_task():
    connection = sqlite3.connect('todo_list.sqlite')
    cursor = connection.cursor()

    task_id = request.forms.get('task_id')
    values = { 'task_id' : task_id }
    query = 'DELETE FROM Task WHERE task_id = :task_id'
    cursor.execute(query, values)

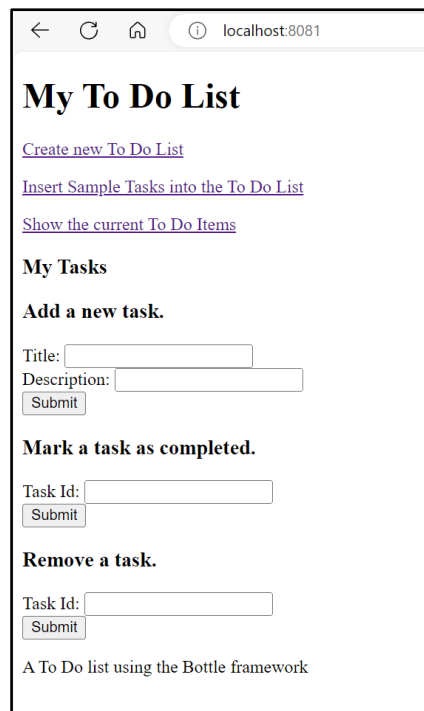
    connection.commit()
    connection.close()

    return template('index')
```

3. Go to the **index.tpl** file
4. Add the following section of code below the section to add a new task to the to do list:

```
<section>
    <form action="/remove_task" method="post">
        <h3>Remove a task.</h3>
        Task Id: <input type="text" name="task_id" /><br />
        <input type="submit">
    </form>
</section>
```

5. Once again, make sure the web app is running and reload the page. Your page should look something like this:



← ↻ 🏠 ⓘ localhost:8081

## My To Do List

[Create new To Do List](#)

[Insert Sample Tasks into the To Do List](#)

[Show the current To Do Items](#)

### My Tasks

**Add a new task.**

Title:

Description:

**Mark a task as completed.**

Task Id:

**Remove a task.**

Task Id:

A To Do list using the Bottle framework

- To test your code, add a task and click the **Submit** button. Then click the link to **Show the current To Do Items** and you should see that the task has now been deleted from the list.

## Adding Some Styling

To make our app a bit nicer to look at and easier to use, let's add some basic styling through CSS. To do this we will create an external stylesheet and hook it up to our template. For our template to pick up the stylesheet we will need to create a static folder and add a route to the static files in the main app. We will follow these steps:

- Create a stylesheet in the **static** folder
  - Import the **static\_file** function from Bottle to the main app (as part of the initial **import** statement)
  - Add a new route and function to the main app to link to the static files
1. Create a new file in the **static** folder and name it **style.css**
  2. Add the following CSS to the file you just created:

```
body {
    background-color: lightsteelblue;
}

section {
    border: solid 3px;
    margin: 10px;
    padding: 5px;
}

table, td, th {
    border: solid 1px;
    border-collapse: collapse;
}

h1, footer {
    text-align: center;
}
```

3. Go to the **todo\_app.py** file
4. In the routes section, use the code below to add a new route below the *remove\_task* route:

```
# Static files
@route('/static/<filename>')
def static(filename):
    return static_file(filename, root='./static')
```

7. Go to the **index.tpl** file
8. Add the following line of code to the **<head>** section of code below the **<title>**:

```
<link type="text/css" href="/static/style.css" rel="stylesheet">
```

9. Once again, make sure the web app is running and reload the page. Your page should look something like this:

**My To Do List**

[Create new To Do List](#)

[Insert Sample Tasks into the To Do List](#)

[Show the current To Do Items](#)

**My Tasks**

task_id	title	description	completed
1	Wash car	Take care to car wash and clean the outside	0
2	Maths	Complete exercise 6 on page 231	1
3	Shopping	Buy ingredients to cook spaghetti bolognese for dinner	0

**Add a new task.**

Title:

Description:

**Mark a task as completed.**

Task Id:

**Remove a task.**

Task Id:

A To Do list using the Bottle framework

## Further Discussion

This is a tutorial to demonstrate how to achieve specific tasks using the Python Bottle framework. There are numerous improvements that can be made to the code and further functionality that is needed to make this a viable product.

Some ideas for further improvement include:

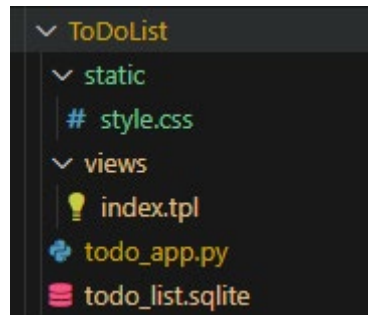
- Increased modularisation to improve the python code. For example, functions should be written to create connections to the database and run the queries.
- Increased feedback to the user. For example, there is no indication that a new table has been created successfully or if there was an error
- Add a constant for the name of the database file so it can be used each time a connection to the database is established

## Appendix: Code Listing

When you have finished working through the tutorial, your code should look something like the code below.

### File Structure

You should have the following folders and files:



### todo\_app.py

```
from bottle import route, run, template, request, static_file
import sqlite3

#####
#
# Routes for each page
#
# Home page
@route('/')
def index():
    return template('index')

# Create new To Do list (remove existing list and create new empty table)
@route('/create_todo_list')
def create_todo_list():
    connection = sqlite3.connect('todo_list.sqlite')
    cursor = connection.cursor()

    query = 'DROP TABLE IF EXISTS Task'
    cursor.execute(query)

    query = '''
        CREATE TABLE Task
        (task_id INTEGER PRIMARY KEY,
         title TEXT NOT NULL,
         description TEXT,
         completed BOOLEAN DEFAULT FALSE)
    '''
    cursor.execute(query)
```

```
connection.commit()
connection.close()

return template('index')

# Insert some sample data to set up the to do list
@route('/insert_sample_data')
def insert_sample_data():
    connection = sqlite3.connect('todo_list.sqlite')
    cursor = connection.cursor()

    task1 = { 'title' : 'Wash car',
              'description': 'Take care to car wash and clean the outside',
              'completed' : False }
    task2 = { 'title' : 'Maths',
              'description' : 'Complete exercise 6 on page 231',
              'completed' : True }
    task3 = { 'title' : 'Shopping',
              'description' : 'Buy ingredients to cook spaghetti bolognese
for dinner',
              'completed' : False }

    query = 'INSERT INTO Task (title, description, completed) VALUES (:title,
:description, :completed)'
    cursor.execute(query, task1)
    cursor.execute(query, task2)
    cursor.execute(query, task3)

    connection.commit()
    connection.close()

    return template('index')

# Retrieve all the items in the to do list and display them on the page
@route('/show_to_do_list')
def show_to_do_list():
    connection = sqlite3.connect('todo_list.sqlite')
    connection.row_factory = sqlite3.Row
    cursor = connection.cursor()

    query = 'SELECT * FROM Task'
    cursor.execute(query)
    tasks = cursor.fetchall()

    connection.close()

    return template('index', records=tasks)
```

```
# Add new task
@route('/add_task', method='POST')
def add_task():
    connection = sqlite3.connect('todo_list.sqlite')
    cursor = connection.cursor()

    # Retrieve the values from the form as a dictionary
    # NOTE: HTML form field names (from the template) must match parameter
names in INSERT query
    form_data = dict(request.params)
    query = 'INSERT INTO Task (title, description) VALUES (:title,
:description)'
    cursor.execute(query, form_data)

    connection.commit()
    connection.close()

    return template('index')

# Mark task as completed
@route('/complete_task', method='POST')
def complete_task():
    connection = sqlite3.connect('todo_list.sqlite')
    cursor = connection.cursor()

    task_id = request.forms.get('task_id')
    values = { 'task_id' : task_id }
    query = '''
        UPDATE Task
        SET completed = TRUE
        WHERE task_id = :task_id
    '''

    cursor.execute(query, values)

    connection.commit()
    connection.close()

    return template('index')

# Add new task
@route('/remove_task', method='POST')
def remove_task():
    connection = sqlite3.connect('todo_list.sqlite')
    cursor = connection.cursor()

    task_id = request.forms.get('task_id')
    values = { 'task_id' : task_id }
```

```

query = 'DELETE FROM Task WHERE task_id = :task_id'
cursor.execute(query, values)

connection.commit()
connection.close()

return template('index')

# Static files
@route('/static/<filename>')
def static(filename):
    return static_file(filename, root='./static')

#####
#
# Helper functions
#####
#
# Run the web server to serve up the pages
#
run(host='localhost', port=8081, debug=True, reloader=True)

```

## index.tpl

```

<!DOCTYPE html>
<html>
  <head>
    <title>My To Do List</title>
    <link type="text/css" href="/static/style.css" rel="stylesheet">
  </head>
  <body>
    <h1>My To Do List</h1>
    <section>
      <p><a href="/create_todo_list">Create new To Do List</a></p>
    </section>
    <section>
      <p><a href="/insert_sample_data">Insert Sample Tasks into the To
Do List</a></p>
    </section>
    <section>
      <p><a href="/show_to_do_list">Show the current To Do Items</a></p>
      <h3>My Tasks</h3>
      % if defined('records'):
        % if len(records) < 1:
          <p><strong>You have no items in your To Do list</strong></p>
        % else:
          <table>

```



```

        <tr>
            % for field in records[0].keys():
            <th> {{ field }} </th>
            % end
        </tr>
        % for record in records:
        <tr>
            % for field in record:
            <td>{{ field }}</td>
            % end
        </tr>
        % end
    </table>
    % end
</section>
<section>
    <form action="/add_task" method="post">
        <h3>Add a new task.</h3>
        Title: <input type="text" name="title" /><br />
        Description: <input type="text" name="description" /><br />
        <input type="submit">
    </form>
</section>
<section>
    <form action="/complete_task" method="post">
        <h3>Mark a task as completed.</h3>
        Task Id: <input type="text" name="task_id" /><br />
        <input type="submit">
    </form>
</section>
<section>
    <form action="/remove_task" method="post">
        <h3>Remove a task.</h3>
        Task Id: <input type="text" name="task_id" /><br />
        <input type="submit">
    </form>
</section>
<footer>
    <p>A To Do list using the Bottle framework</p>
</footer>
</body>
</html>

```

## style.css

```
body {  
    background-color: lightsteelblue;  
}  
  
section {  
    border: solid 3px;  
    margin: 10px;  
    padding: 5px;  
}  
  
table, td, th {  
    border: solid 1px;  
    border-collapse: collapse;  
}  
  
h1, footer {  
    text-align: center;  
}
```