2/21/2024

Python Flask

What Is it, Tutorial and How to Use It

FELIPE ALCÁZAR GÓMEZ
ENRIQUE SÁNCHEZ-MIGALLÓN OCHOA
ESI UCLM – CIUDAD REAL

Contents

1. What is Python Flask

1.1. What is Flask?

Flask is a lightweight and flexible **web framework** for Python. It is designed to start with web development in a quick and easy way, with the ability to **scale up** to complex applications as needed.

We can highlight the following characteristics regarding Flask:

- **Minimalistic**: provides only the essentials needed to get a web application up and running.
- **Modular**: allowing developers to easily add or remove components as required by their application.
- **Extensible**: developers can easily integrate third-party extensions to add additional functionality to their applications.

1.2. WSGI

WSGI is a specification for a universal interface between the web server and the web applications. Enabling developers to build powerful and scalable web applications in Python.

1.3. Werkzeug

It is a WSGI toolkit, which implements requests, response objects, and other utility functions. This enables building a web framework on top of it.

1.4. Jinja2

It allows developers to generate dynamic content by embedding Python code within HTML templates.

2. Uses of Python Flask

Flask provides tools, libraries, patterns for building web applications, managing HTTP requests, sessions, integration with databases and other web technologies. Important to remark that there are many standard libraries that enhance the overall functionality when using python flask such as: numpy, pandas, matplot…

3. How to Install

3.1. Prerequisites

- Windows 10/11 – Ubuntu – MacOS.
- Python 2.6 or higher (Python 3 recommended but the best is Python 2.7).

3.2. Install virtualenv and Flask

It is recommended to create **virtual development environment** to avoid compatibility issues with different versions. Here are the commands to create it:

- *> pip install virtualenv*
  or
  > sudo apt-get install virtualenv
- *> mkdir flaskproject*
- *> cd flaskproject*
- *> virtualenv venv*

To use the virtual environment, we can get inside it using this command:

- *> venv/bin/activate*

Once inside the virtual environment, we must install **flask** with this command:

- *> pip install flask*

3.3. Using Visual Studio Code

Visual Studio Code is a powerful editor that simplifies writing code and using Flask. It has many extensions, but we recommend the following one:

https://marketplace.visualstudio.com/items?itemName=cstrap.flask-snippets

Then, we must get inside the virtual environment on Visual Studio Code and then create a new python file and we can start programming.

4. First steps: My First Program

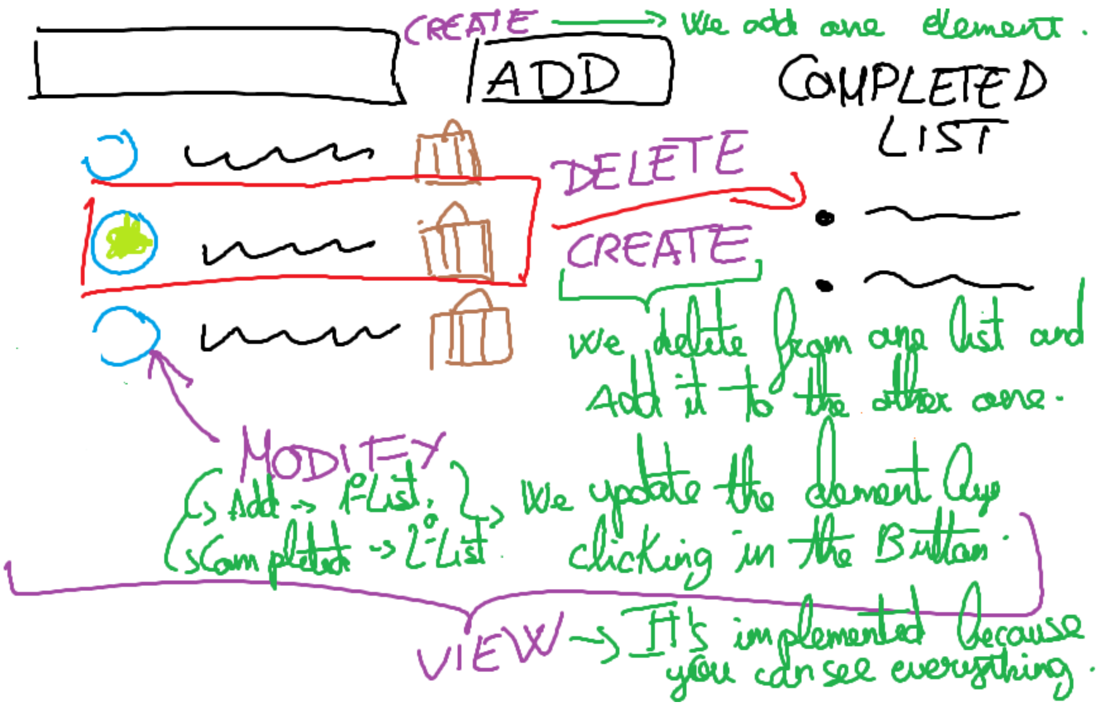
4.1. GitHub Repository.

Here we have uploaded the code.

https://github.com/Enriquesmo/DSW-FlaskTutorial.git

4.2. Sketch.

First, we create a Sketch that will guide us in the future. This is a brainstorming of the functionalities that we want to implement. The result is the following:

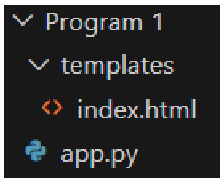

CREATE → We add one element.
ADD
COMPLETED LIST
DELETE
CREATE
We delete from one list and Add it to the other one.
MODIFY
(→ Add → 1º List,)
(→ Completed → 2º List.
We update the element by clicking in the Button.
VIEW → It's implemented because you can see everything.

Summarizing, our goal is to create a webpage that implements the functionality of a checklist by means of an input and some buttons. There will be 3 different programs, from a simple one that only includes the tasks being added, to a more detailed one that allows us to remove or undone tasks.

4.3. 1º Upgrade. Add Bar and List.

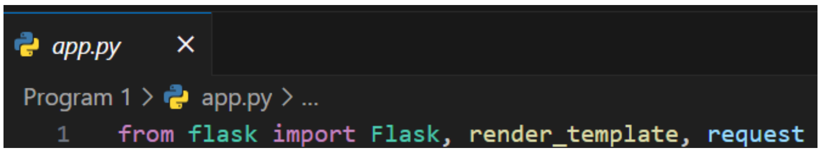
In this First version of the program, we implement the functionalities of the "Add Bar", the "Add Button" and the "added elements List".

First, we must create 2 files and 1 directory. The directory will be called "templates" that will hold the .html files (This folder is compulsory in all flask programs). Then, we create the first file called "index.html" that will be in the templates directory and will contain the interface. Finally, we create the last file called "app.py" that will contain the functionalities.



```
Program 1
  templates
    index.html
  app.py
```

Now let's explain the **app.py** file in detail. On this portion of code, we include Flask, render_template (which allows us to render the HTML files with new data) and request (to obtain the objects of the HTML file):

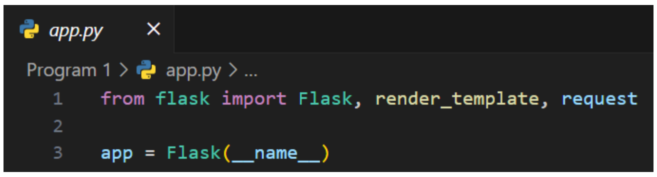


app.py

Program 1 > app.py > ...

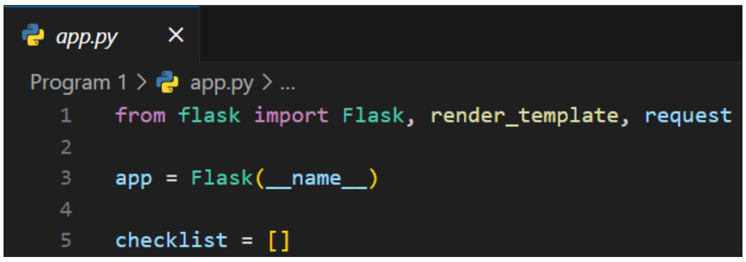
```python
from flask import Flask, render_template, request
```

Then, we create the Flask web server in the line 2. The variable "app" its now the web application and will be used to run the server and interact with other parts of the code:



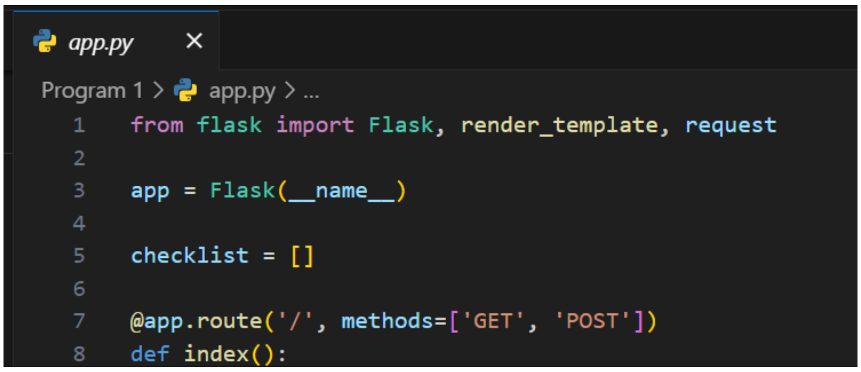
```python
from flask import Flask, render_template, request

app = Flask(__name__)
```

After that, an array called “checklist” is defined so that all elements that are being created can be stored:



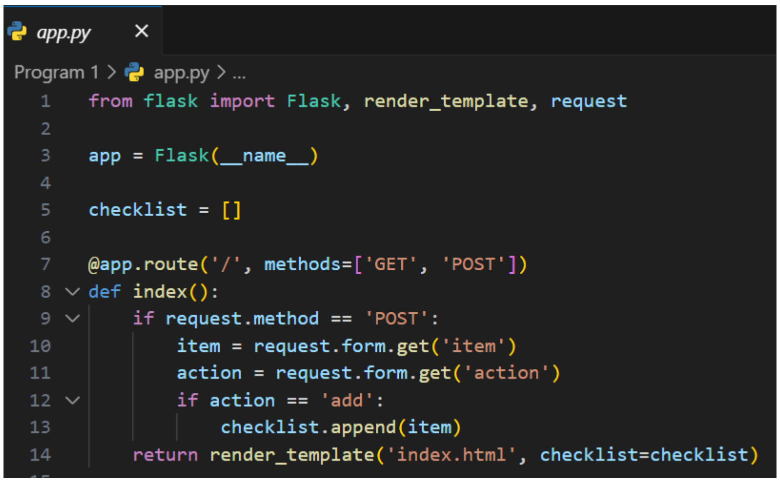
```python
from flask import Flask, render_template, request

app = Flask(__name__)

checklist = []
```

Now, the important function that will make the program work. By means of the app variable and its decorator (that flask provides to route web requests) we can perform operations inside the function “index” whenever a client does a GET or POST request on the root page (indicated by “/”):



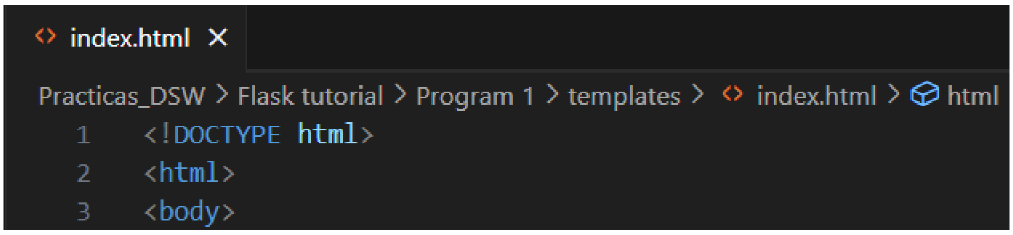
```python
from flask import Flask, render_template, request

app = Flask(__name__)

checklist = []

@app.route('/', methods=['GET', 'POST'])
def index():
```

The index function returns the checklist updated and the page index.html by means of render_template (which as we said earlier, updates the html file). Before that, we have an if statement to identify the request being performed (this request variable is given by flask and allows us to identify different actions). In our case, we want to know when a POST action is performed. In this example, the post action Is the adding of an element so that, when it happens, we add it to the list of elements (checklist variable). This can be done thanks to retrieving the data from the html (using request.form.get() function):



```python
from flask import Flask, render_template, request

app = Flask(__name__)

checklist = []

@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        item = request.form.get('item')
        action = request.form.get('action')
        if action == 'add':
            checklist.append(item)
    return render_template('index.html', checklist=checklist)
```

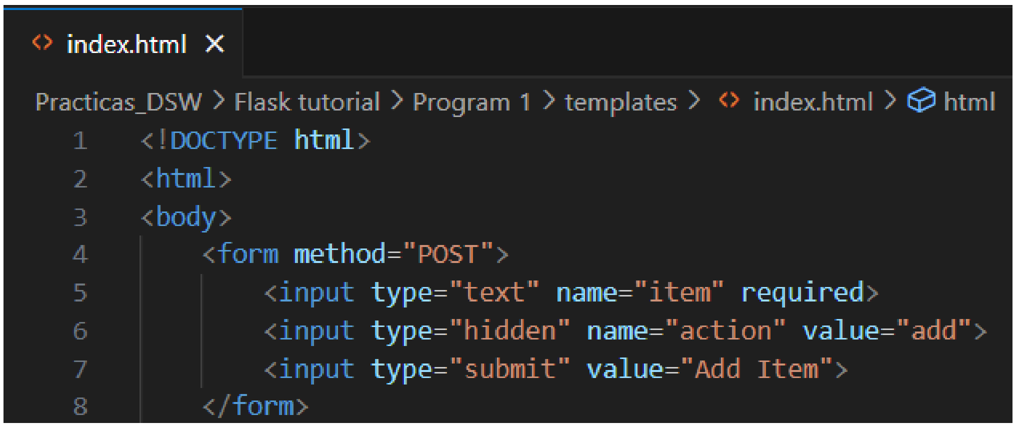
Now, we will explain the **index.html** file in detail. In this portion of code, we define the document type and we start it:


index.html ×
Practicas_DSW > Flask tutorial > Program 1 > templates > index.html > html

```html
<!DOCTYPE html>
<html>
<body>
```

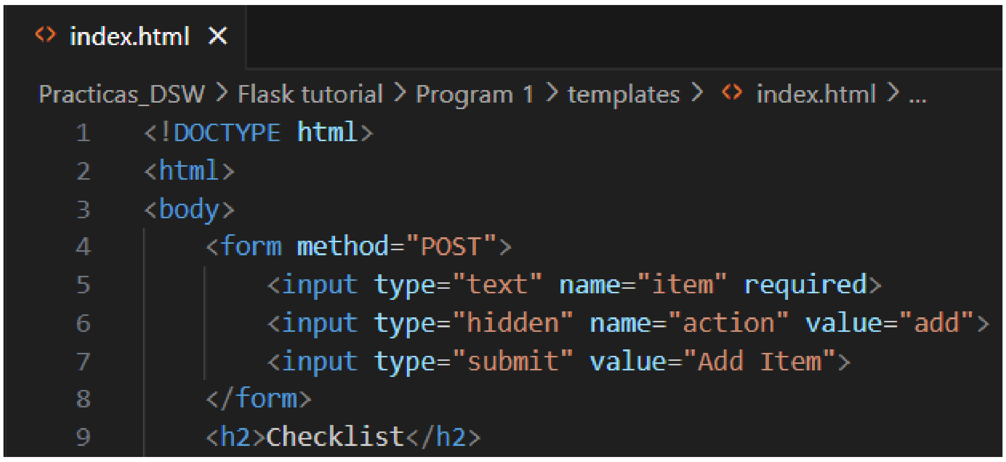
Then, we create a HTML form to add new items to the checklist.

- **<form method="POST">** This form, when submitted, sends a POST request.
- **<input type="text" name="item" required>** This is a text input field where users can enter a new item for the checklist. The *required* attribute means that the form cannot be submitted unless this field has a value.
- **<input type="hidden" name="action" value="add">** This is a hidden field that has a constant value of "add". When the POST request is sent to the server, the server will consult this field to determinate what to do. This is useful to control the action being taken on the app.py file.
- **<input type="submit" value="Add Item">** This is the submit button for the form. When clicked, it submits the form. The text on the button is "Add Item".


index.html ×
Practicas_DSW > Flask tutorial > Program 1 > templates > index.html > html

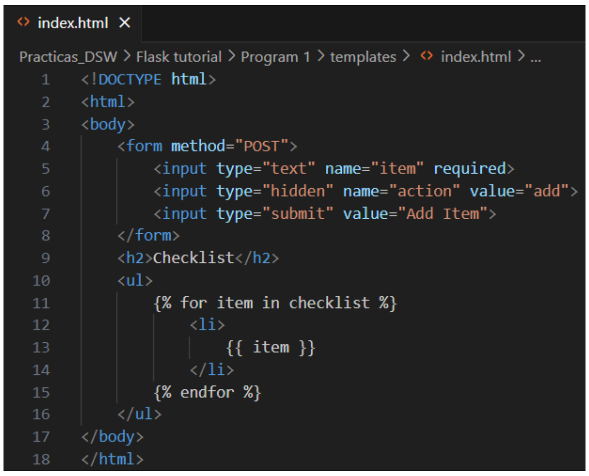
```html
<!DOCTYPE html>
<html>
<body>
    <form method="POST">
        <input type="text" name="item" required>
        <input type="hidden" name="action" value="add">
        <input type="submit" value="Add Item">
    </form>
```

Now, we display a text line which contains the words “Checklist”.


index.html ×
Practicas_DSW > Flask tutorial > Program 1 > templates > index.html > ...

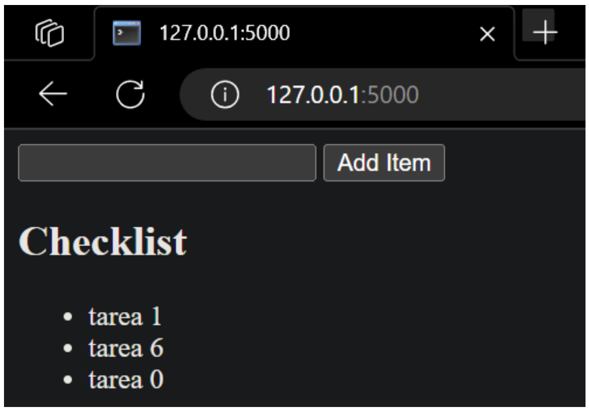
```html
<!DOCTYPE html>
<html>
<body>
    <form method="POST">
        <input type="text" name="item" required>
        <input type="hidden" name="action" value="add">
        <input type="submit" value="Add Item">
    </form>
    <h2>Checklist</h2>
```

Finally, we display the list called “Checklist” that we created in the “app.py” file. First, we define an unordered list with **<ul>**. Then we use a **JINJA2 FOR LOOP**. This allows us to implement a JavaScript function in a html file. We can access the **checklist** variable because we are using flask. Finally, for each loop, we add one of the elements of the list in the html.


index.html
Practicas_DSW > Flask tutorial > Program 1 > templates > index.html > ...

```
<!DOCTYPE html>
<html>
<body>
    <form method="POST">
        <input type="text" name="item" required>
        <input type="hidden" name="action" value="add">
        <input type="submit" value="Add Item">
    </form>
    <h2>Checklist</h2>
    <ul>
        {% for item in checklist %}
            <li>
                {{ item }}
            </li>
        {% endfor %}
    </ul>
</body>
</html>
```

So, when a user enters text into the text field and clicks "Add Item", a POST request is sent with the text of the new item and an action indicating that the item should be added to the checklist. This is then handled in the app.py file. The result can be appreciated in this image:



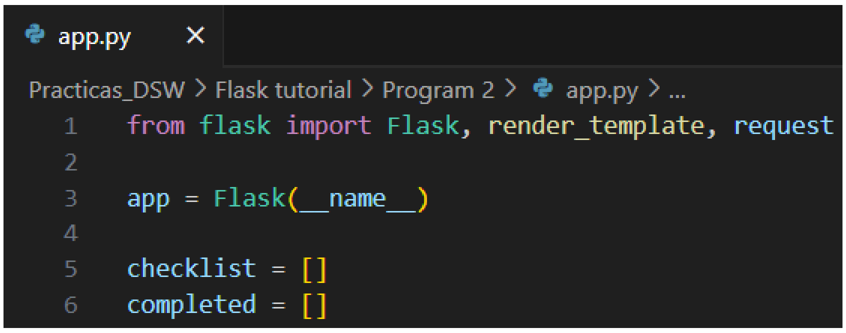
Checklist

- tarea 1
- tarea 6
- tarea 0

4.4. 2º Upgrade. Completed List.

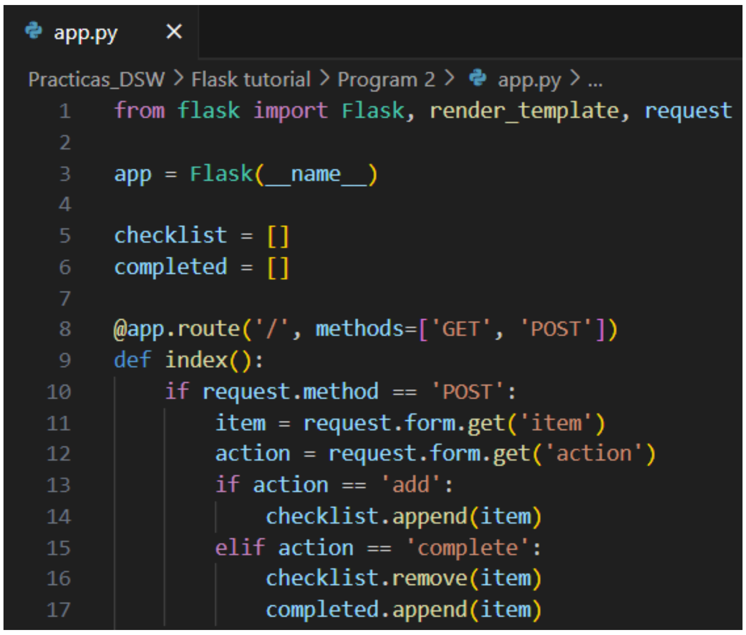
In this second program, we are going to add the functionality of the second list called “Completed”. Now, each the elements of the Checklist, will have a new button called “**Done**” that will remove it from the 1º Checklist and add it to the “**Completed**” list.

First, we are going to explain the “**app.py**” file. We can see that we added a new variable called “completed” that is a list that will contain the removed elements of the “checklist” list:


app.py
Practicas_DSW > Flask tutorial > Program 2 > app.py > ...

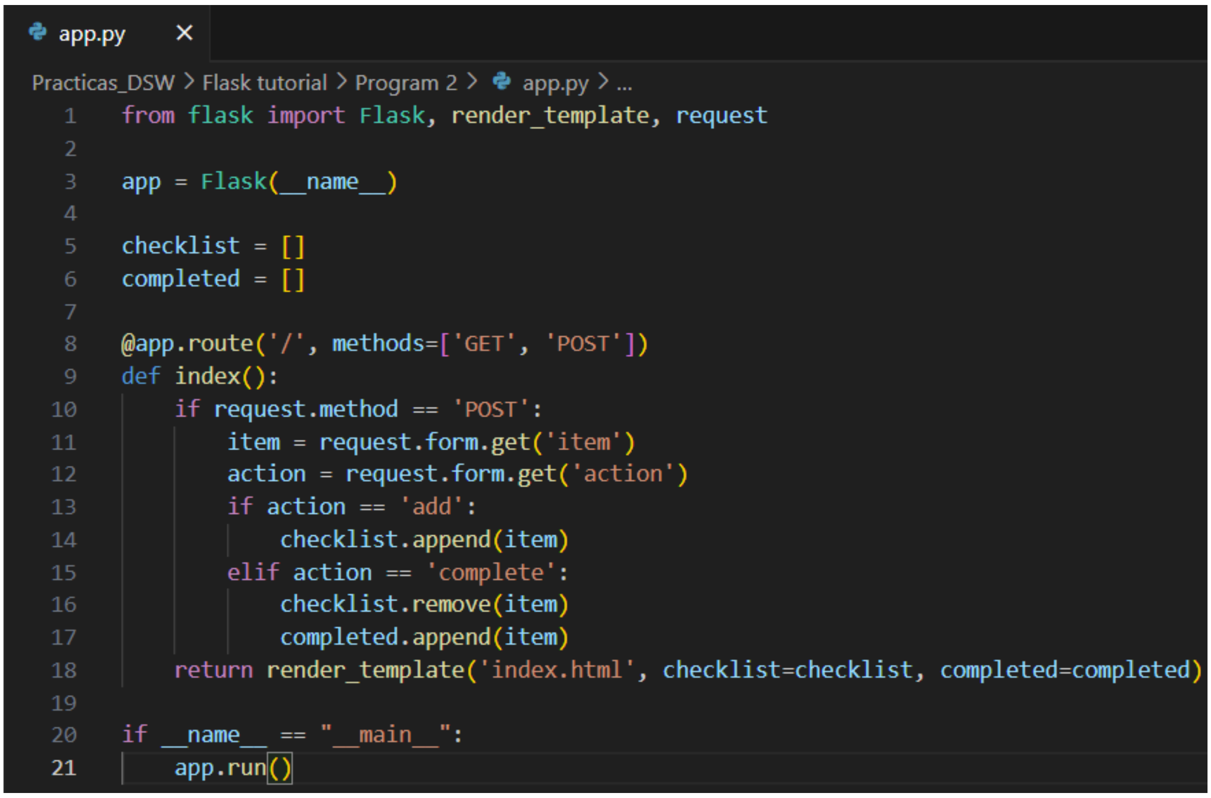
```
from flask import Flask, render_template, request

app = Flask(__name__)

checklist = []
completed = []
```

Now, we have added an “elif” that check if the action contents is “complete”. If it is, we remove the item from the checklist and add it to the completed list:



```python
from flask import Flask, render_template, request

app = Flask(__name__)

checklist = []
completed = []

@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        item = request.form.get('item')
        action = request.form.get('action')
        if action == 'add':
            checklist.append(item)
        elif action == 'complete':
            checklist.remove(item)
            completed.append(item)
```

Finally, in the “**return render_template**”, we must return the early created “completed” list to the html.

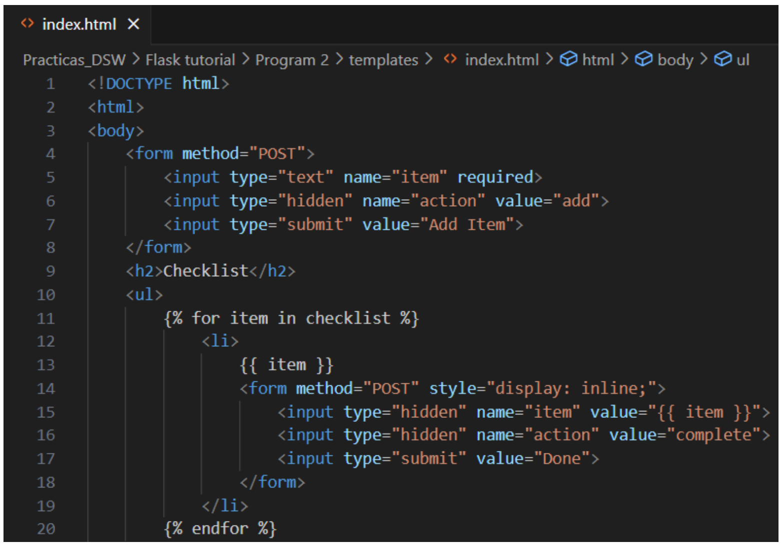


```python
from flask import Flask, render_template, request

app = Flask(__name__)

checklist = []
completed = []

@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        item = request.form.get('item')
        action = request.form.get('action')
        if action == 'add':
            checklist.append(item)
        elif action == 'complete':
            checklist.remove(item)
            completed.append(item)
    return render_template('index.html', checklist=checklist, completed=completed)

if __name__ == "__main__":
    app.run()
```

Now, we are going to explain the changes in the “**index.html**” file.

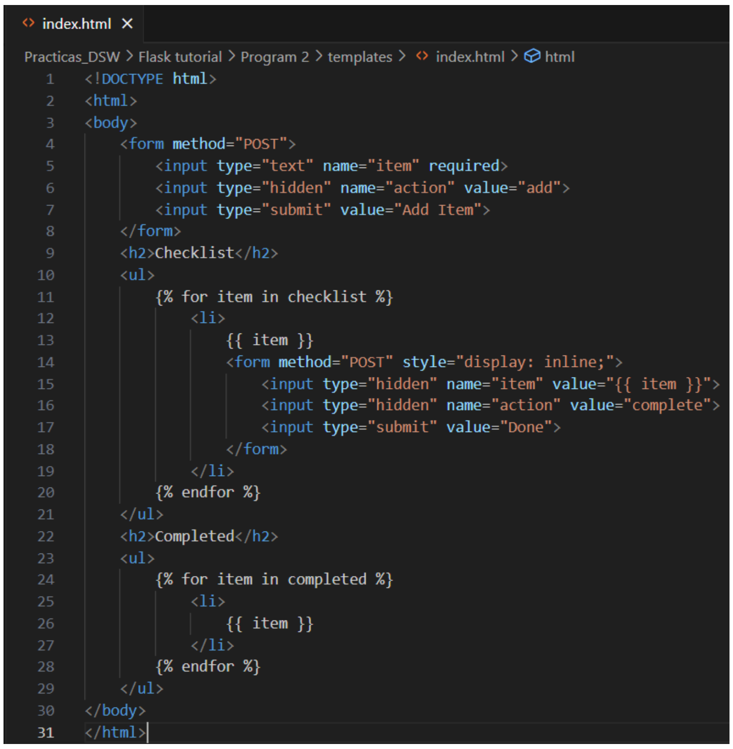
The first change we can see is the new form added in each element of the Checklist. This form is used to mark that item and send it to the server to process it:

- **<form method="POST" style="display: inline;">** This form, when submitted, sends a POST request. The `style="display: inline;"` attribute makes the form display inline with other elements.
- **<input type="hidden" name="item" value="{{ item }}">** This is a hidden input field that has a value of the current item in the checklist (from the Jinja2 variable `{{ item }}`).
- **<input type="hidden" name="action" value="complete">** This is another hidden input field that has a constant value of "complete". When the POST request is sent to the server, the server will consult this field to determinate what to do. This is useful to control the action being taken on the app.py file.
- **<input type="submit" value="Done">** This is the submit button for the form. When clicked, it submits the form. The text on the button is "Done".


index.html
Practicas_DSW > Flask tutorial > Program 2 > templates > index.html > html > body > ul

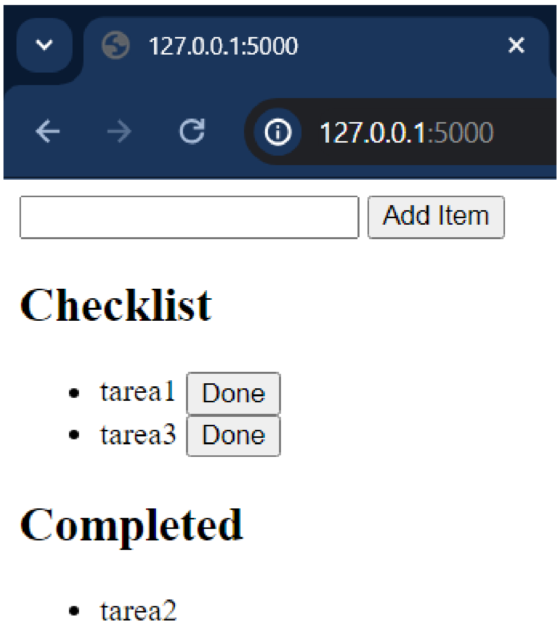
```html
<!DOCTYPE html>
<html>
<body>
    <form method="POST">
        <input type="text" name="item" required>
        <input type="hidden" name="action" value="add">
        <input type="submit" value="Add Item">
    </form>
    <h2>Checklist</h2>
    <ul>
        {% for item in checklist %}
            <li>
                {{ item }}
                <form method="POST" style="display: inline;">
                    <input type="hidden" name="item" value="{{ item }}">
                    <input type="hidden" name="action" value="complete">
                    <input type="submit" value="Done">
                </form>
            </li>
        {% endfor %}
```

Finally, we display a text that contains the words “Completed” and then we display the items of the completed list, as we did in the last Upgrade with the Checklist:


index.html
Practicas_DSW > Flask tutorial > Program 2 > templates > index.html > html

```html
<!DOCTYPE html>
<html>
<body>
    <form method="POST">
        <input type="text" name="item" required>
        <input type="hidden" name="action" value="add">
        <input type="submit" value="Add Item">
    </form>
    <h2>Checklist</h2>
    <ul>
        {% for item in checklist %}
            <li>
                {{ item }}
                <form method="POST" style="display: inline;">
                    <input type="hidden" name="item" value="{{ item }}">
                    <input type="hidden" name="action" value="complete">
                    <input type="submit" value="Done">
                </form>
            </li>
        {% endfor %}
    </ul>
    <h2>Completed</h2>
    <ul>
        {% for item in completed %}
            <li>
                {{ item }}
            </li>
        {% endfor %}
    </ul>
</body>
</html>
```

So, when a user clicks "Done", a POST request is sent with the name of the item to be marked as complete and an action indicating that the item should be marked as complete.



Checklist

- tarea1 Done
- tarea3 Done

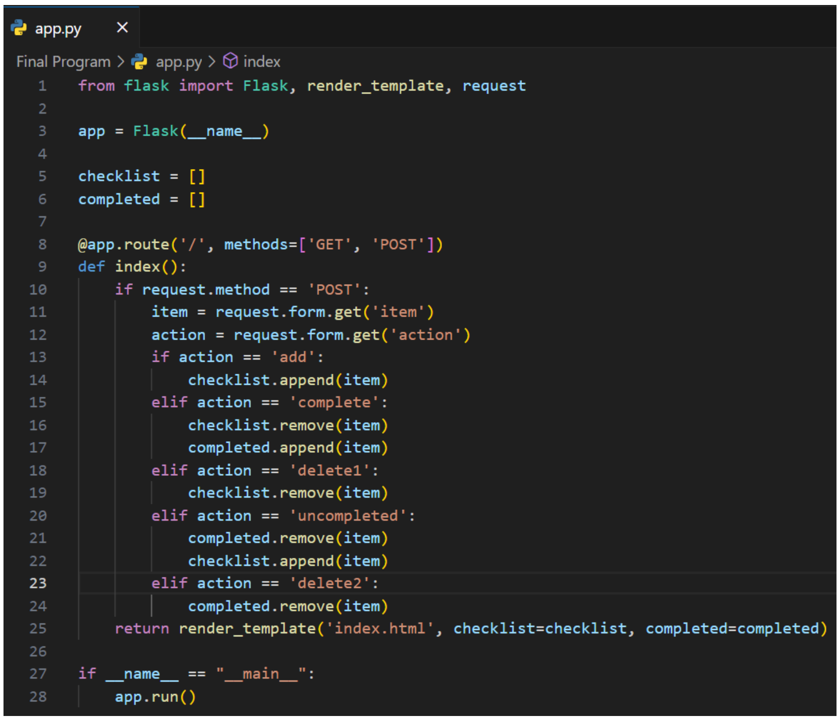
Completed

- tarea2

4.5. 3º Upgrade. Extra functionalities.

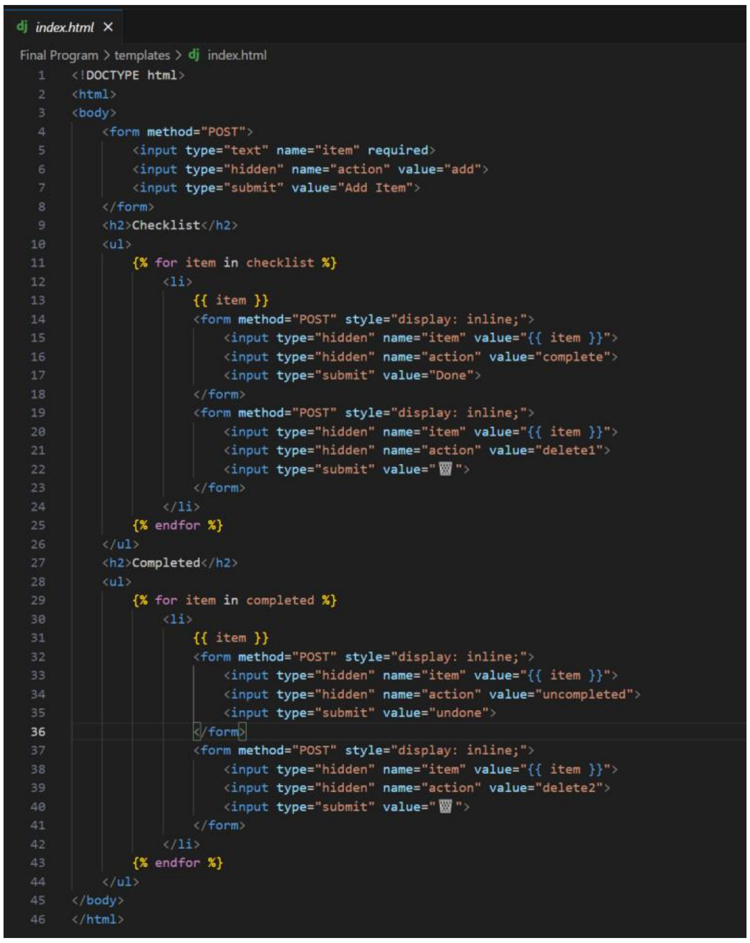
On this final program our goal is to be able to remove and mark as “done” all the tasks that have been created and those completed.

To achieve our goal, we have added new actions in the index.html and appropriately handled them inside the **app.py**. We have these new actions: delete1, uncompleted (referring to undo a completed task) and delete2 (referring to delete a completed task). Their purpose can be guessed from their names, since they only remove or append items from the lists depending on the action performed:



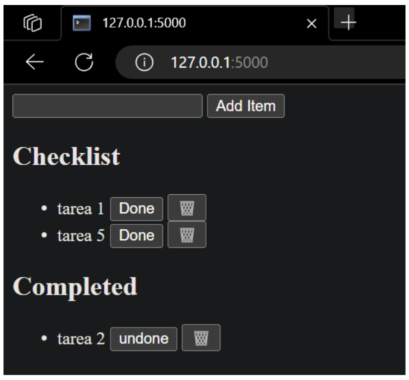
```python
from flask import Flask, render_template, request

app = Flask(__name__)

checklist = []
completed = []

@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        item = request.form.get('item')
        action = request.form.get('action')
        if action == 'add':
            checklist.append(item)
        elif action == 'complete':
            checklist.remove(item)
            completed.append(item)
        elif action == 'delete1':
            checklist.remove(item)
        elif action == 'uncompleted':
            completed.remove(item)
            checklist.append(item)
        elif action == 'delete2':
            completed.remove(item)
    return render_template('index.html', checklist=checklist, completed=completed)

if __name__ == "__main__":
    app.run()
```

Once we know that, let's see what has changed in our index.html file. As we have seen in previous changes, we have added 3 new forms that are used to extract information from different actions. In this case, the buttons undone, and a thrash icon (in ASCII) are included per task:



```html
<!DOCTYPE html>
<html>
<body>
    <form method="POST">
        <input type="text" name="item" required>
        <input type="hidden" name="action" value="add">
        <input type="submit" value="Add Item">
    </form>
    <h2>Checklist</h2>
    <ul>
        {% for item in checklist %}
            <li>
                {{ item }}
                <form method="POST" style="display: inline;">
                    <input type="hidden" name="item" value="{{ item }}">
                    <input type="hidden" name="action" value="complete">
                    <input type="submit" value="Done">
                </form>
                <form method="POST" style="display: inline;">
                    <input type="hidden" name="item" value="{{ item }}">
                    <input type="hidden" name="action" value="delete1">
                    <input type="submit" value="🗑">
                </form>
            </li>
        {% endfor %}
    </ul>
    <h2>Completed</h2>
    <ul>
        {% for item in completed %}
            <li>
                {{ item }}
                <form method="POST" style="display: inline;">
                    <input type="hidden" name="item" value="{{ item }}">
                    <input type="hidden" name="action" value="uncompleted">
                    <input type="submit" value="undone">
                </form>
                <form method="POST" style="display: inline;">
                    <input type="hidden" name="item" value="{{ item }}">
                    <input type="hidden" name="action" value="delete2">
                    <input type="submit" value="🗑">
                </form>
            </li>
        {% endfor %}
    </ul>
</body>
</html>
```

The final result of the program is the following:



```
127.0.0.1:5000

[          ] Add Item

Checklist

  • tarea 1 Done 🗑
  • tarea 5 Done 🗑

Completed

  • tarea 2 undone 🗑
```

5. Bibliography

- Tutorialspoint - Flask quickguide
- Flask (web framework) - Wikipedia
- Visual Studio 2022 IDE - Programming Tool for Software Developers (microsoft.com)