# Server-side (Python)

First of all, all python is run through `app.py`. However you can create external python scripts and just run the `import script_name`.

The only things you need to know for this is the `@app.route()` command. How this works is whenever anyone visits the website, for example, if the domain is `solvexchange.com` and you have a command for `@app.route('/forum')`, if the user visits the domain `solvexchange.com/forum` it will run the function which is stored immediately below it. An example of this being implemented is as follows:

```
@app.route('/forum')
def render_forum():
    return render_template('forum.html') # This function essentially
just opens the html page, nothing special.
```

`render_template()` has two inputs, first is a string containing the html page's filename. These html files MUST be placed in the `templates` folder. The second input is used for rendering by passing in parameters.

```
@app.route('/forum')
def render_forum():
    return render_template('forum.html', some_var=some_var) # This will
make it so that where ever there is a {{ some_var }} in the render
template it will be replaced by the values stored in some_var
```

There are three methods but only two are primarily used. The three are `POST`, `GET` and `DELETE`. `POST` is when you send data from the client-side to the server, `GET` is when you receive data from the server. `DELETE` is when you remove something from the client.

For some of functions, only one of them is needed. For example, if you create a refresh function that would not refresh the page. Only `GET` would be needed, however, if you would want to send some data to only receive data for a certain user, you may need to use `POST` as well.

## Render Templates

Render templates are used to render things from the server. How these work is by rendering using variables. These render templates do not have the traditional html body and head, they instead use `{{% block head|body %}}` and `{{% endblock %}}`. To insert variables for example, if you had a a string containing an username which is unique to each user called `user`. You would just add `{{ user }}` and it would show what is stored in the variable `user`. For lists, you can do for loops so that each have styling. `{% for item in list %}`, item can be named anything, it works similarly to how python's `for i in list:` works.

# Client-side (JavaScript)

To send stuff to the server, you must run specific functions within the JavaScript.

```
function SendMessage() {
    $.ajax({
        url: '/destination', // Put whatever is inside the @app.route()
        type: 'POST', // This should remain the same
        contentType: 'application/json',
        data: JSON.stringify({ 'email' : emailInput.value }),
        success: function(response) {
// Run whatever once it works correctly
// response is a json which can hold data from the server.
// response.result should return if it worked or not
        },
        error: function(xhr, status, error) {
            console.error('Error: ', error); // error handling
        }
    });
}
```

When doing this, `POST` can also get values from the server when it returns the response from the server. This can be done when returning values from the server in `app.py` by just adding your values within a list then running `jsonify(list)` then returning said value. It will make it a dictionary however but this should be fine if you just give it a key like `jsonify('list_name' : list)`.