Jinja

- The folks behind Flask are also the developers of a popular *templating engine* called **Jinja**.

- The primary motivation for a templating engine like this is for us to be able to procedurally generate HTML based on the value of some variable(s) in our programs.

- This allows us to mix Python and HTML!

- Let’s build a simple web application to create a multiplication table, where the size of the table is determined by the user, and we generate the HTML for the table based on the size the user wants.

- We will be making use of Flask’s `render_template()` method rather extensively in this application.

- First, the basic app.

- First, the basic app.

```
from flask import Flask, render_template, request
app = Flask(__name__)

@app.route("/")
def mult_table():
```

- Let’s start by just displaying a simple form to the user.

```
from flask import Flask, render_template, request
app = Flask(__name__)

@app.route("/")
def mult_table():
```

- Let’s start by just displaying a simple form to the user.

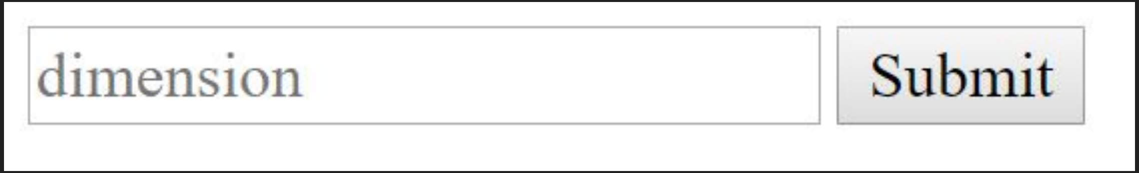
```
from flask import Flask, render_template, request
app = Flask(__name__)

@app.route("/")
def mult_table():
    return render_template("form.html")
```

- By default, Flask will look in the `templates/` directory to try to find a template with that name, so first we create that subdirectory, and then we toss a very simple form in there. (No Jinja in this one, since it’s static.)

```html
<!DOCTYPE html>
<html>
    <head>
        <title>
            Multiplication Table
        </title>
    </head>
    <body>
        <form action="/" method="post">
            <input name="size" type="number" placeholder="dimension"/>
            <input name="submit" type="submit" />
        </form>
    </body>
</html>
```

```html
<!DOCTYPE html>
<html>
    <head>
        <title>
            Multiplication Table
        </title>
    </head>
    <body>
        <form action="/" method="post">
            <input name="size" type="number" placeholder="dimension"/>
            <input name="submit" type="submit" />
        </form>
    </body>
</html>
```

```html
<!DOCTYPE html>
<html>
    <head>
        <title>
            Multiplication Table
        </title>
    </head>
    <body>
        <form action="/" method="post">
            <input name="size" type="number" placeholder="dimension"/>
            <input name="submit" type="submit" />
        </form>
    </body>
</html>
```



- Okay, so now we’re good on displaying the form. But what about when the user **submits** the form? Right now, the form just keeps refreshing.

```
from flask import Flask, render_template, request
app = Flask(__name__)

@app.route("/")
def mult_table():
    return render_template("form.html")
```

- Okay, so now we're good on displaying the form. But what about when the user **submits** the form? Right now, the form just keeps refreshing.

```
from flask import Flask, render_template, request
app = Flask(__name__)

@app.route("/", methods=["GET", "POST"])
def mult_table():
    return render_template("form.html")
```

- Okay, so now we’re good on displaying the form. But what about when the user **submits** the form? Right now, the form just keeps refreshing.

```
from flask import Flask, render_template, request
app = Flask(__name__)

@app.route("/", methods=["GET", "POST"])
def mult_table():
    if request.method == "GET":
        return render_template("form.html")
```

- Okay, so now we’re good on displaying the form. But what about when the user **submits** the form? Right now, the form just keeps refreshing.

```
from flask import Flask, render_template, request
app = Flask(__name__)

@app.route("/", methods=["GET", "POST"])
def mult_table():
    if request.method == "GET":
        return render_template("form.html")
    # our form is set up to submit via POST
    elif request.method == "POST":
        return render_template("table.html")
```

- Time to create another template. We know that HTML tables consist of <tr> tags for each row, consisting of a set of <td> tags for columns. So that lets us craft the super-basic idea for a template.

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Table</title>
  </head>
  <body>
        <table>

          <tr>

              <td>

              </td>

          </tr>

        </table>
  </body>
</html>
```

- Time to create another template. We know that HTML tables consist of <tr> tags for each row, consisting of a set of <td> tags for columns. So that lets us craft the super-basic idea for a template.

- Next, we need to somehow convey to this template the number of rows the user supplied. We can do this by altering our call to `render_template()`.

- Okay, so now we’re good on displaying the form. But what about when the user **submits** the form? Right now, the form just keeps refreshing.

```
from flask import Flask, render_template, request
app = Flask(__name__)

@app.route("/", methods=["GET", "POST"])
def mult_table():
    if request.method == "GET":
        return render_template("form.html")
    # our form is set up to submit via POST
    elif request.method == "POST":
        return render_template("table.html")
```

- Okay, so now we’re good on displaying the form. But what about when the user **submits** the form? Right now, the form just keeps refreshing.

```
from flask import Flask, render_template, request
app = Flask(__name__)

@app.route("/", methods=["GET", "POST"])
def mult_table():
    if request.method == "GET":
        return render_template("form.html")
    # our form is set up to submit via POST
    elif request.method == "POST":
        return render_template("table.html", dim=request.form.get("size"))
```

- Okay, so now we’re good on displaying the form. But what about when the user **submits** the form? Right now, the form just keeps refreshing.

```
from flask import Flask, render_template, request
app = Flask(__name__)

@app.route("/", methods=["GET", "POST"])
def mult_table():
    if request.method == "GET":
        return render_template("form.html")
    # our form is set up to submit via POST
    elif request.method == "POST":
        return render_template("table.html", dim=request.form.get("size"))
```

- Time to create another template. We know that HTML tables consist of <tr> tags for each row, consisting of a set of <td> tags for columns. So that lets us craft the super-basic idea for a template.

- Next, we need to somehow convey to this template the number of rows the user supplied. We can do this by altering our call to `render_template()`.

- Effectively, “dim” is now a variable within Jinja, and Jinja allows us to use a Python-like syntax interspersed within our HTML.

- Jinja is introduced in the template in one of two ways:
  - `{% ... %}`
    - These delimiters indicate that what is between them is control-flow or logic.
  - `{{ … }}`
    - These delimiters indicate that what is between them should be evaluated and effectively “printed” as HTML.

- Exactly what you can do with Jinja is an exercise for home, but its syntax is generally Python like, with a couple of quirks due to the way it is interspersed with HTML. Let’s see how we can use it to generate the HTML we need.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Table</title>
  </head>
  <body>
        <table>

          <tr>

              <td>

              </td>

          </tr>

        </table>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Table</title>
  </head>
  <body>
        <table>
        // loop to repeat “dim” times (“dim” # of rows)
          <tr>
        // loop to repeat “dim” times (“dim” # of columns)
              <td>
        // print out that value of the cell between <td>s
              </td>

          </tr>

        </table>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Table</title>
  </head>
  <body>
        <table>
          {% for i in range(dim) %}
          <tr>
        // loop to repeat “dim” times (“dim” # of columns)
              <td>
        // print out that value of the cell between <td>s
              </td>

          </tr>
          {% endfor %}
        </table>
  </body>
</html>
```

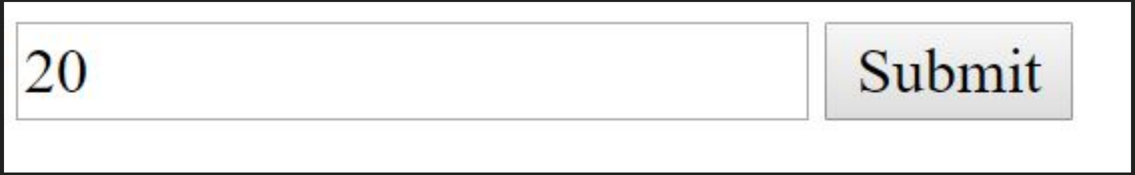
```
<!DOCTYPE html>
<html>
  <head>
    <title>Table</title>
  </head>
  <body>
        <table>
          {% for i in range(dim) %}
          <tr>
              {% for j in range(dim) %}
              <td>
        // print out that value of the cell between <td>s
              </td>
              {% endfor %}
          </tr>
          {% endfor %}
        </table>
  </body>
</html>
```

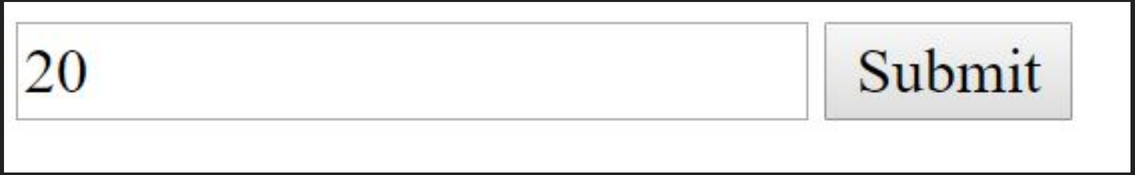
```
<!DOCTYPE html>
<html>
  <head>
    <title>Table</title>
  </head>
  <body>
        <table>
          {% for i in range(dim) %}
          <tr>
              {% for j in range(dim) %}
              <td>
                 {{ (i + 1) * (j + 1) }}
              </td>
              {% endfor %}
          </tr>
          {% endfor %}
        </table>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Table</title>
  </head>
  <body>
        <table>
          {% for i in range(dim) %}
          <tr>
              {% for j in range(dim) %}
              <td>
                  {{ (i + 1) * (j + 1) }}
              </td>
              {% endfor %}
          </tr>
          {% endfor %}
        </table>
  </body>
</html>
```

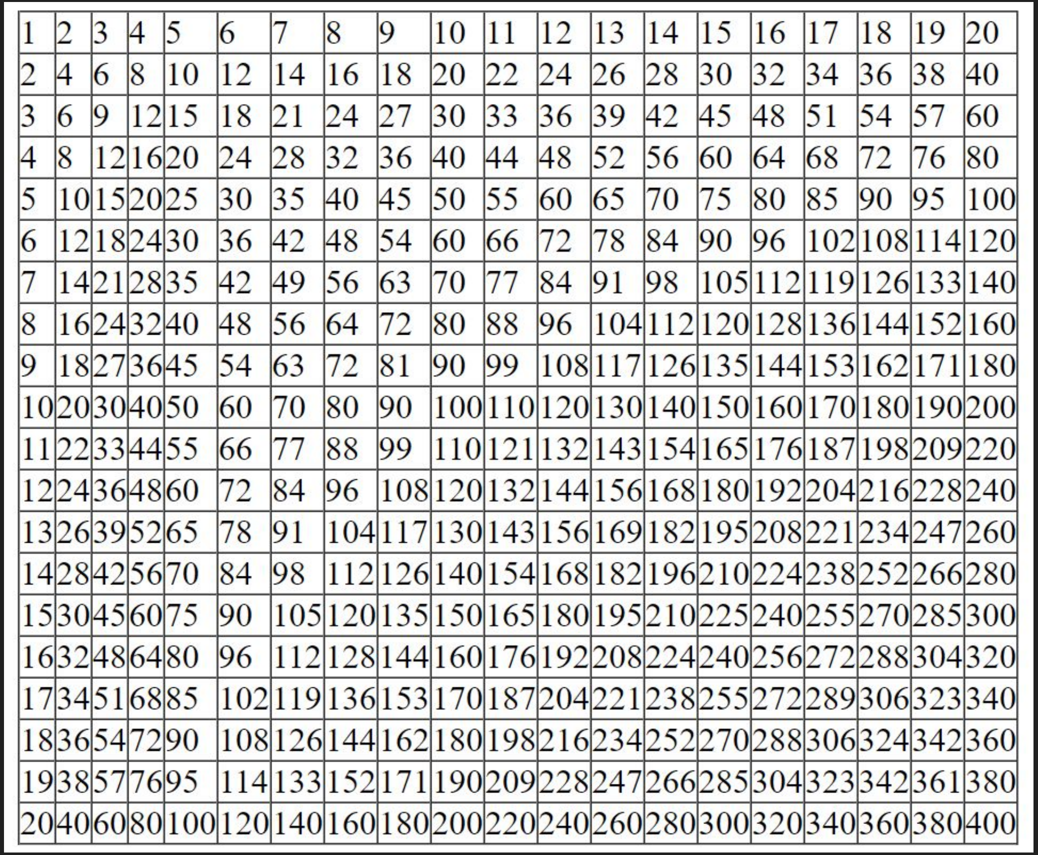
```
<!DOCTYPE html>
<html>
  <head>
    <title>Table</title>
  </head>
  <body>
        <table>
          {% for i in range(dim|int) %}
          <tr>
              {% for j in range(dim|int) %}
              <td>
                  {{ (i + 1) * (j + 1) }}
              </td>
              {% endfor %}
          </tr>
          {% endfor %}
        </table>
  </body>
</html>
```

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Table</title>
  </head>
  <body>
        <table border=1>
          {% for i in range(dim|int) %}
          <tr>
              {% for j in range(dim|int) %}
              <td>
                  {{ (i + 1) * (j + 1) }}
              </td>
              {% endfor %}
          </tr>
          {% endfor %}
        </table>
  </body>
</html>
```

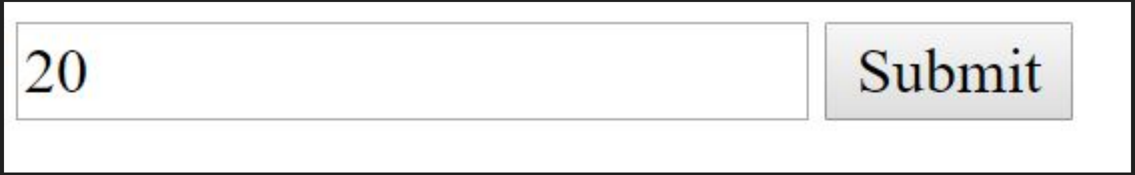

20
Submit

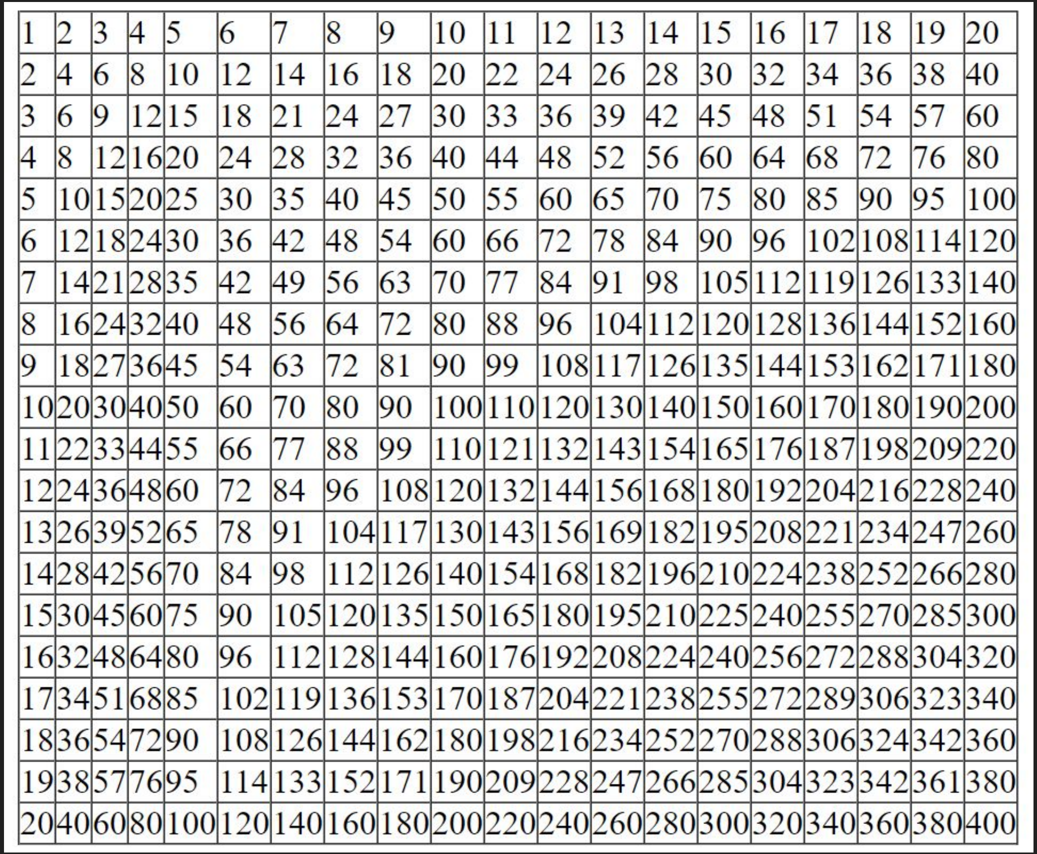


20



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
3	6	9	12	15	18	21	24	27	30	33	36	39	42	45	48	51	54	57	60
4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	68	72	76	80
5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
6	12	18	24	30	36	42	48	54	60	66	72	78	84	90	96	102	108	114	120
7	14	21	28	35	42	49	56	63	70	77	84	91	98	105	112	119	126	133	140
8	16	24	32	40	48	56	64	72	80	88	96	104	112	120	128	136	144	152	160
9	18	27	36	45	54	63	72	81	90	99	108	117	126	135	144	153	162	171	180
10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200
11	22	33	44	55	66	77	88	99	110	121	132	143	154	165	176	187	198	209	220
12	24	36	48	60	72	84	96	108	120	132	144	156	168	180	192	204	216	228	240
13	26	39	52	65	78	91	104	117	130	143	156	169	182	195	208	221	234	247	260
14	28	42	56	70	84	98	112	126	140	154	168	182	196	210	224	238	252	266	280
15	30	45	60	75	90	105	120	135	150	165	180	195	210	225	240	255	270	285	300
16	32	48	64	80	96	112	128	144	160	176	192	208	224	240	256	272	288	304	320
17	34	51	68	85	102	119	136	153	170	187	204	221	238	255	272	289	306	323	340
18	36	54	72	90	108	126	144	162	180	198	216	234	252	270	288	306	324	342	360
19	38	57	76	95	114	133	152	171	190	209	228	247	266	285	304	323	342	361	380
20	40	60	80	100	120	140	160	180	200	220	240	260	280	300	320	340	360	380	400





1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
3	6	9	12	15	18	21	24	27	30	33	36	39	42	45	48	51	54	57	60
4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	68	72	76	80
5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
6	12	18	24	30	36	42	48	54	60	66	72	78	84	90	96	102	108	114	120
7	14	21	28	35	42	49	56	63	70	77	84	91	98	105	112	119	126	133	140
8	16	24	32	40	48	56	64	72	80	88	96	104	112	120	128	136	144	152	160
9	18	27	36	45	54	63	72	81	90	99	108	117	126	135	144	153	162	171	180
10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200
11	22	33	44	55	66	77	88	99	110	121	132	143	154	165	176	187	198	209	220
12	24	36	48	60	72	84	96	108	120	132	144	156	168	180	192	204	216	228	240
13	26	39	52	65	78	91	104	117	130	143	156	169	182	195	208	221	234	247	260
14	28	42	56	70	84	98	112	126	140	154	168	182	196	210	224	238	252	266	280
15	30	45	60	75	90	105	120	135	150	165	180	195	210	225	240	255	270	285	300
16	32	48	64	80	96	112	128	144	160	176	192	208	224	240	256	272	288	304	320
17	34	51	68	85	102	119	136	153	170	187	204	221	238	255	272	289	306	323	340
18	36	54	72	90	108	126	144	162	180	198	216	234	252	270	288	306	324	342	360
19	38	57	76	95	114	133	152	171	190	209	228	247	266	285	304	323	342	361	380
20	40	60	80	100	120	140	160	180	200	220	240	260	280	300	320	340	360	380	400

It may not look beautiful… but that's what CSS is for!