Consider a table with fields “name”, “address”, “id”, and “favorite_pokemon”.

1. Using db.execute, insert a row with these values:

    - Name: Phyllis
    - Address: Sleepy st.
    - ID: 270
    - Favorite_pokemon: mudkip

2. Using db.execute, delete all rows with favorite_pokemon pikachu.

3. Using db.execute, select all rows with ID greater than or equal to 270.

4. Save the rows you just selected in a variable called winner.

5. What syntax do we use to incorporate python in our HTML files?

6. How might you use this particular syntax to incorporate “winner” as a table in our HTML file?

CS50 Section 9

```
<html>
    <head>
        <title>
            Current Time
        </title>
    </head>
    <body>
        The current time is 10:58
    </body>
</html>
```

```
<html>
    <head>
        <title>
            Current Time
        </title>
    </head>
    <body>
        The current time is 10:59
    </body>
</html>
```

```
<html>
    <head>
        <title>
            Current Time
        </title>
    </head>
    <body>
        The current time is 11:00
    </body>
</html>
```

- Pure HTML is static; the only way to update our page's content is to manually edit the source files. Incorporating Python can make our code so much more flexible.

```
from flask import Flask
from datetime import datetime
from pytz import timezone

app = Flask(__name__)

@app.route("/")
def time():
    now = datetime.now(timezone('America/New_York'))
    return f"The current date and time is {now}."
```

- It’s rather easy to get started using Flask within CS50 IDE.

```
from flask import Flask
```

- It’s rather easy to get started using Flask within CS50 IDE.

```
from flask import Flask
```

- After importing the Flask module, we need to initiate a Flask application.

```
app = Flask(__name__)
```

- It’s rather easy to get started using Flask within CS50 IDE.

```
from flask import Flask
```

- After importing the Flask module, we need to initiate a Flask application.

```
app = Flask(__name__)
```

- From there, we need only write functions to define the behavior of our application.

```
def time():
```

```
def index():
    return "You are at the index page!"


def sample():
    return "You are on the sample page!"
```

```
@app.route("/")
def index():
    return "You are at the index page!"

@app.route("/sample")
def sample():
    return "You are on the sample page!"
```

- Data can be passed in via HTML forms as well, as via HTTP POST, but we need to clarify things a bit more for Flask if we do:

```
@app.route("/login", methods=['GET', 'POST'])
def login():
    # if the username field of form missing
    if not request.form.get("username"):
        return apology("need a username!")
```

- Or we could vary our function's behavior depending on what kind of HTTP request we got.

```
@app.route("/login", methods=['GET', 'POST'])
def login():
    if request.method == "POST":
        # do one thing
    else:
        # do a different thing
```

- Flask has a number of different built-in methods you'll find useful.

redirect()

session()

render_template()

JINJA

- 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():
        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.

form.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():

    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")
```

form.html

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.

table.html

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

       <tr>

          <td>

          </td>

       </tr>

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

- 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", dim=request.form.get("size"))
```

- 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.

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

Lab: Birthdays

PSET: Finance

- Start early!
- Review use of session in Flask
  - Get user id: session[“user_id”]
  - Clear any user id (i.e. when no user is logged in): session.clear()

Test

- Week 1: C Basics
- Week 2: Compiling, command line args, assembly code
- Week 3: Run time / Complexity analysis, sorting algorithms, recursion
- Week 4: Pointers, memory, file reading
- Week 5: Data structures: linked lists, BSTs, hash tables, stacks, queues
- Week 6: Python syntax
- Week 7: SQL
- Week 8: IP/TCP/HTTP
- Week 9: Flask

Red: definitely review. Blue: Less important, could brush up on

Test

Practice Questions: https://cs50.harvard.edu/college/2020/fall/test/

Review sheets: https://www.dropbox.com/sh/5y662ey1hc4sde4/AACjgHN3NtSKk4ShsRDFd_Sja?dl=0

CS50 Shorts:

https://www.youtube.com/playlist?list=PLhQjrBD2T381k8ul4WQ8SQ165XqY149WW

Test Tips

- Actually study :P
- Regular test-taking skills apply
  - Time management
  - Answer every question
  - Be concise!
- If you're asked to write code, use the IDE!
- May ask heads@cs50.harvard.edu questions, but not anyone else
- START EARLY
  - Within a day or two of release: skim questions, make a plan
- Good luck! :)