In the beginning, I suspect this was Greek to a large number of you.
Over the semester, we have visited every one of these nodes, from the JavaScript running on the front end client to our three tier architecture, to databases.

Adapted from Armando Fox and David Patterson (Berkeley cs169) under CC-BY-SA-NC license.
Some of the Good Parts of JavaScript

```javascript
const wrapValue = (n) => { // function(n) {
  let local = n;
  return () => local; // function () { return local; }
}
const wrap1 = wrapValue(1);
const wrap2 = wrapValue(2);
console.log(wrap1()); // What will print here?
console.log(wrap2()); // What will print here?
```

And we had to figure out new syntax (arrow functions!) and new semantics (closures!)
Then you learned the Tao of React.
const movies = new Map();

app.get('/api/movies', (request, response) => {
  response.send(Array.from(movies.values()));
});

app.get('/api/movies/:id', (request, response) => {
  response.send(movies.get(parseInt(request.params.id, 10)));
});

app.put('/api/movies/:id', (request, response) => {
  const movieId = parseInt(request.params.id, 10);
  const newMovie = request.body;
  const mergedMovie = Object.assign({}, movies.get(movieId), newMovie);
  movies.set(mergedMovie.id, mergedMovie);
  response.send(mergedMovie);
});

<table>
<thead>
<tr>
<th>Route</th>
<th>Controller Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST /movies</td>
<td>Create new movie from request data</td>
</tr>
<tr>
<td>GET /movies/:id</td>
<td>Read data of movie with id == :id</td>
</tr>
<tr>
<td>PUT /movies/:id</td>
<td>Update movie with id == :id from request data</td>
</tr>
<tr>
<td>DELETE /movies/:id</td>
<td>Delete movie with id == :id</td>
</tr>
<tr>
<td>GET /movies</td>
<td>List (read) all movies</td>
</tr>
</tbody>
</table>

We learned about writing backend servers using Node to create RESTful APIs
We learned about two different approaches to data persistence on the backend in the form of relational databases and so-called NoSQL databases.

<table>
<thead>
<tr>
<th></th>
<th>Relational (RDBMS)</th>
<th>Non-Relational</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data</strong></td>
<td>Table-oriented</td>
<td>Document-oriented, key-value, graph-based, column-oriented, ...</td>
</tr>
<tr>
<td><strong>Schema</strong></td>
<td>Fixed schema</td>
<td>Dynamic schema</td>
</tr>
<tr>
<td><strong>Joins</strong></td>
<td>Used extensively</td>
<td>Used infrequently</td>
</tr>
<tr>
<td><strong>Interface</strong></td>
<td>SQL</td>
<td>Custom query language</td>
</tr>
<tr>
<td><strong>Transactions</strong></td>
<td>ACID</td>
<td>CAP</td>
</tr>
</tbody>
</table>

```
SELECT * FROM people
WHERE age > 25;
```

```
db.people.find(
    { age: { $gt: 25 } }
)
```
We learned about test driven development and concepts like unit testing and integration testing. We also talked about the use of behavior driven development as a way to move us from design to implementation through test.

Google testing blog

Kent C Dodds “Write tests. Not too many. Mostly integration.”
We learned about using git to manage development, and the value of continuous integration.

https://www.atlassian.com/git/tutorials/using-branches
You also learned about ways to approach design from user stories to CRC cards, to lo-fi prototypes.
Agile development processes, Scrum in particular, played an important role for us.

Adapted from Mountain Goat Software
And we thought about how to approach building our own Mono Lisas.
Over the course of the semester, you have done four assignments and 12 practicals yielding up a fairly complex wikipedia-like platform with database persistence and user authentication.

Between these and the project, we have created 824 repositories on github. Just on the projects, we have made 1753 commits across just the projects (roll-race has a comfortable lead with 382 commits, more than 100 more than the next closest) and done 2307 Travis builds.
And the 40 different acronyms....
Take-aways

- Behind every design decision there should be a user story
- Testing, not just a class requirement, it’s a good idea
- Develop iteratively
- There should be one source of truth
- Don’t repeat yourself
- Don’t mutate state
- Read error messages
- Automate all of the things
- Don’t break the build
- Avoid code smells
Write beautiful code, do the Right Time and give the Statue of Liberty hair