SQL and JS Pitfalls

Assignment 2 Preparation
SQL Concepts
SQL vs. NoSQL

- [https://www.youtube.com/watch?v=Nu1UQblRQdM](https://www.youtube.com/watch?v=Nu1UQblRQdM)
- **SQL**
  - Support for complex aggregation tools
  - Normalized Data
- **NoSQL**
  - Flexible data models
  - Denormalized data
  - Eventual consistency (Inconsistent reading)
SELECT and FROM

● SELECT - What attributes do I want?
  ○ SELECT * - All attributes
  ○ SELECT x, y, z - Column attributes from the table

● FROM - What table (data source) do I want?
Common Postgres Statements

- List all databases: `\list`
- Connecting to a database: `\c database_name`
- List all the tables: `\dt`
- Show the schema for a table: `\d+ table_name`
- Show me everything from the table: `SELECT * FROM TABLE_NAME`
Primary and Foreign Key

- **Primary Key** - The identifier for this model.
  - There is at most ONE primary key per table.
- **Foreign Key** - A reference to an identifier from a different model.
Pitfalls of Javascript
What is Javascript callback?
A callback is a function you call at a certain time. Different methods might offer multiple callbacks or just one and they’ll fire at different times in response to different parts of the original functions.
Callback Example: Animation

- The original “function” in this case is animate, and it offers one place for a callback. Which is when the animation completes. In a case like this where it’s just one function that I am only going to call in this place.

- Once a function is called something is going to happen, so let’s start with something simple, just log a message to the console. So, this animation will run. It’ll animate these properties in this amount of time. And when it finishes, it will fire this message.
Callback Example: Animation

cogs121
What does this code output?

```javascript
for(var i = 0; i < 5; i++){
    setTimeout(function(){
        console.log(i);
    }, 0);
}
```
Callback Hell

- What is callback hell?

```javascript
var somedb;
var someWebAPI;
var token;
var security;

someWebAPI.get('/login', function (req, res) {
  token.checkToken(req.token, function (safe) {
    somedb.find(username, function (req.user) {
      security.checkPassword(user.password, req.password, function (matched) {
        token.createToken(user.id, function (userData) {
          res.send(userData, function () {
            // error
            }, function () {
              // success
          })
        })
      })
    })
  })

  someWebAPI.get('/login')
  .then(token.checkToken(req.token))
  .then(somedb.find(username))
  .then(security.checkPassword(user.password, req.password))
  .then(res.send(userData))
  .success(function () {}) //error
  .error(function () {});
});
```
Callback Hell

- **Note:** This is not executable code. This is only pseudo-code to demonstrate the issue.
- We have four objects. And each of these objects has a function that takes a callback as one of its parameters. And therefore we begin to chain the calls to these objects within the callbacks. We end up with something very unpleasant and confusing to look at. This is a very common problem when dealing with asynchronous code and node. There are several solutions to it. One of them is promises.
What are Promises?

- Promises: return an object which promises to do some work
- This object has separate callbacks for `resolve` and `reject`
- This allows us to work with asynchronous code in a much more synchronous way.
- We want because traditional asynchronous code can lead to a large set of nested callbacks sometimes called `callback hell`
- Other libraries:
  - Async - Asynchronous control flow
  - Q - Tool for creating Promises
  - https://www.youtube.com/watch?v=g90irqWEqd8 - Awesome tutorial -- I Promise
== vs. ===

- ==
  - Performs type conversion, then compares equality

- ===
  - Does not perform any type conversion

- Examples
  - 0 == false
  - 0 === false
  - 'Brian' === new String('Brian')
Tricky Arithmetic

- [https://www.destroyallsoftware.com/talks/wat](https://www.destroyallsoftware.com/talks/wat)
- `||`
  - 0 || 2, 1 || 2
- `&&`
  - 2 && 1, 0 && 3
Questions?