Your Next Week

Saturday March 21

9 AM
— DUE Class 02 Code Challenge
— DUE Class 02 Lab
— DUE Class 03 Reading
— Class 03

MIDNIGHT
— DUE Class 03 Learning Journal

Sunday March 22

MIDNIGHT
— DUE Career: Professional Etiquette
— DUE Class 02 - 03 Feedback

Monday March 23

Tuesday March 24

6:30 PM
— DUE Class 03 Code Challenge
— DUE Class 03 Lab
— DUE Class 04 Reading
— Class 04A

Wednesday March 25

6:30 PM
— Class 04B

MIDNIGHT
— DUE Class 04 Learning Journal

Thursday March 26

6:30 PM
— Lab 04 & Code Challenge 04 Co-Working

Friday March 27

Saturday March 28

9 AM
— DUE ALL PRE-WORK
— DUE Class 04 Code Challenge
— DUE Class 04 Lab
— DUE Class 05 Reading
— Class 05

MIDNIGHT
— DUE Class 04 Learning Journal

Class 03

Data Modeling & NoSQL Databases

seattle-javascript-401n16

Lab 02 Review

Code Challenge 02 Review

Databases

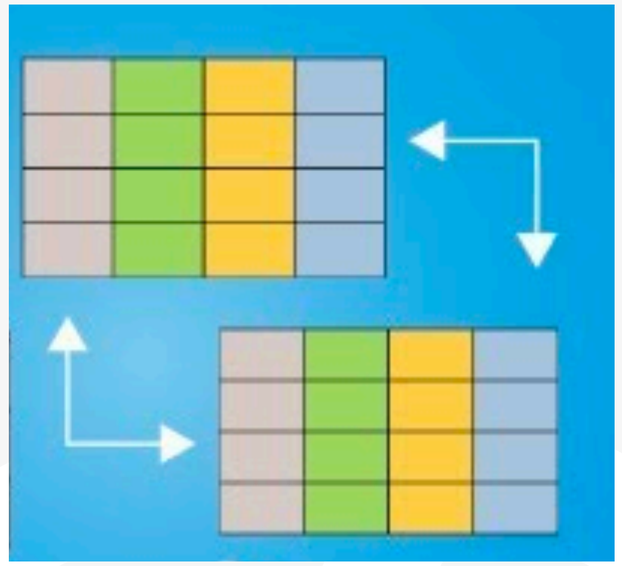
- So far, your notes application lets you:
    - “Add” a note (log it to the console)
    - Validate your command line arguments
    - Validate your created note
- Now we want to save these notes!
- Our application will send information to a database to store long term

Aw CRUD!

- When we connect to a database, there are **four major actions** we want to do
  - **Create** an entry
  - **Read** a stored entry
  - **Update** a stored entry
  - **Delete** a stored entry
- We refer to these basic actions as “**CRUD**”

SQL vs NoSQL

- Postgres is a **relational** database using **structured query language (SQL)**
  - Schema leads to creation of tables
  - If two things are related, another table needs to relate them
- We’re moving to MongoDB, a **non-relational**, **non-SQL (NoSQL)** database
  - Very fluid / flexible
  - Essential JSON data, connections are defined by us



Our Data

```
const noteSchema = {
    id: { type: 'string', required: true },
    note: { type: 'string', required: true },
    categoryId: { type: 'string', required: false },
};
```

```
let myNote = {
    id: '001',
    note: 'Hello World!',
    categoryId: ['001'],
};
```

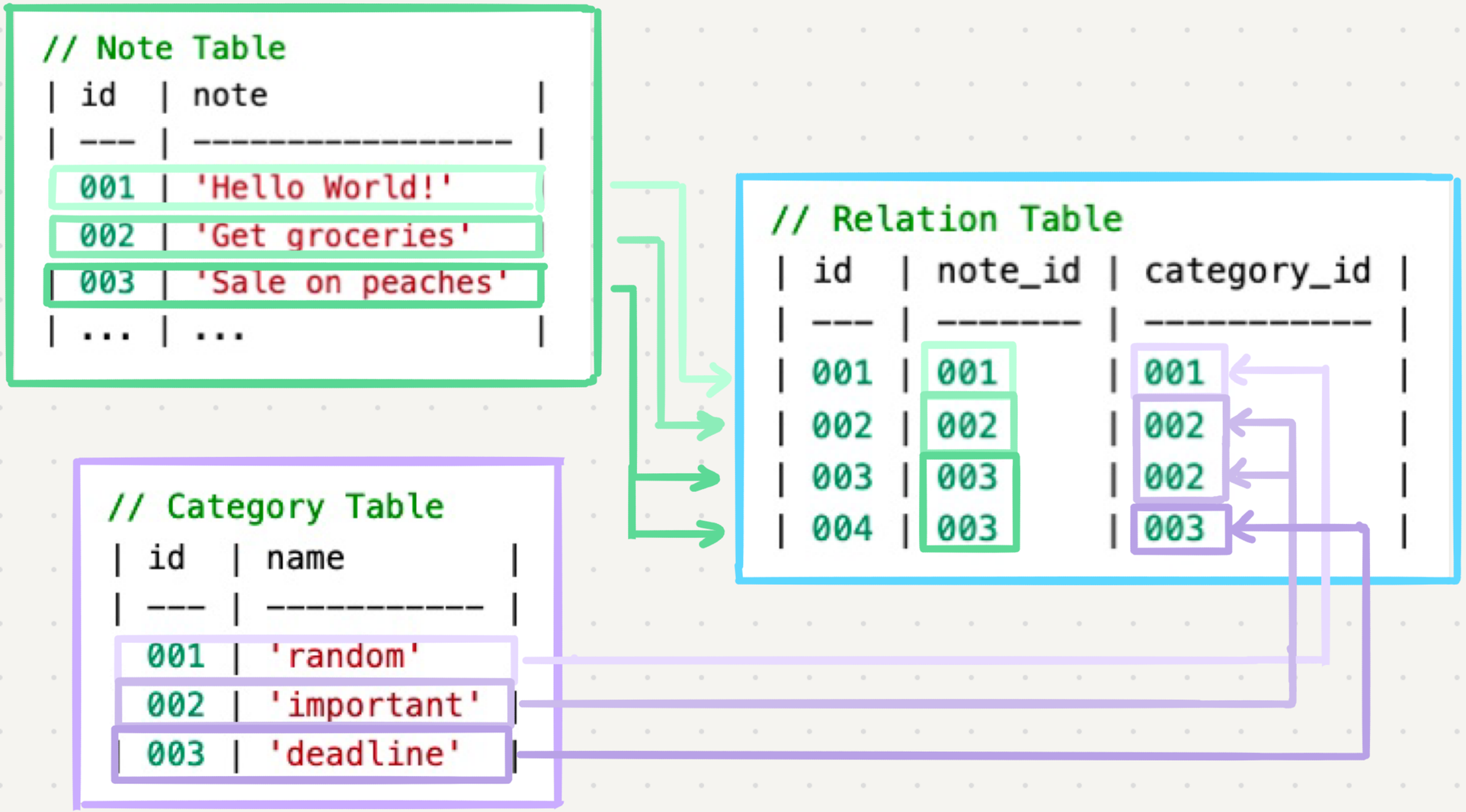
RELATION

```
let categoryRandom = {
    id: '001',
    name: 'random',
};
```

```
const categorySchema = {
    id: { type: 'string', required: true },
    name: { type: 'string', required: true },
};
```

```
let categoryImportant = {
    id: '002',
    name: 'important',
};
```

Represented in SQL



Represented in NoSQL

```
// Notes
[
    {
        id: '001',
        note: 'Hello World!',
        categoryId: ['001'],
    },
    {
        id: '002',
        note: 'Get groceries',
        categoryId: ['002'],
    },
    {
        id: '003',
        note: 'Sale on peaches',
        categoryId: ['002', '003'],
    },
];
```

NO ENFORCED RELATION

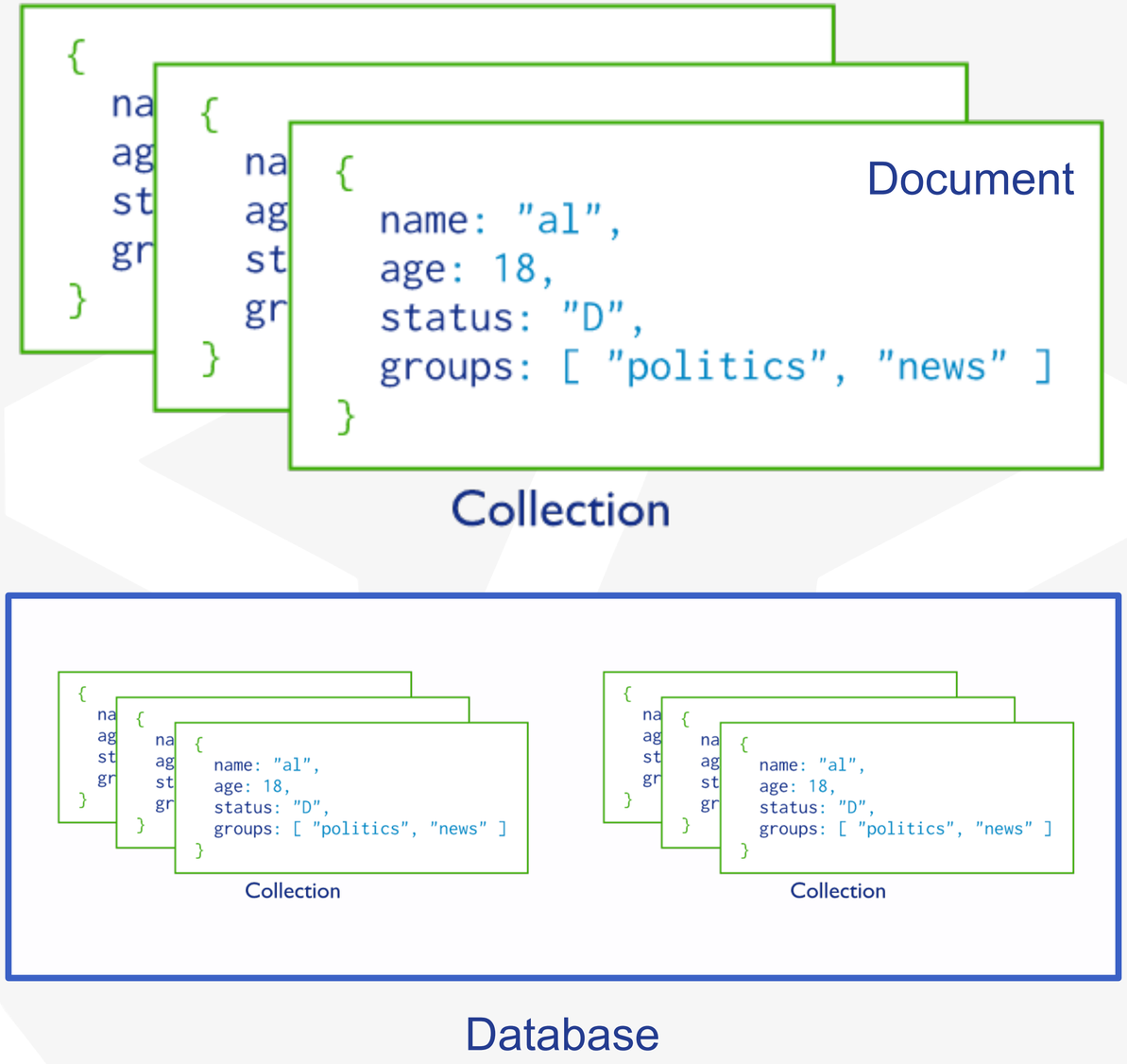
```
// Categories
[
    {
        id: '001',
        name: 'random',
    },
    {
        id: '002',
        name: 'important',
    },
    {
        id: '003',
        name: 'deadline',
    },
];
```

MongoDB

- A very popular NoSQL database system
    - Flexible, scaleable, easy to setup
- Works really well with JavaScript
- Dependency: `npm install mongodb`
- API with many operations
    - More than just CRUD
- Can make both local and “production” databases

Collections and Documents

- MongoDB has new terminology:
  - **Record** - An object representing a piece of data (aka document)
  - **Document** - An individual record for a given collection, stored as a `.bson` (binary JSON) file (aka record)
  - **Collection** - All the records for a specific schema (notes, categories)
  - **Database** - What contains all your data / all your collections



Using MongoDB from CLI

Running Local MongoDB

```
mongod --dbpath=[/PATH/TO/DATA/FOLDER]
```

MongoDB Shell Commands

Command	Description
`mongo`	Launch the mongo shell. Once in the shell, you should see `>`
`show dbs`	Show all the databases
`use db <name>`	Use the database with name `<name>`
`show collections`	Show all the collections in the current database
`db.<collection>.find()`	List all the documents / records in the specified collection `<collection>`
`db.<collection>.save()`	Save a new document / record to the specified collection `<collection>`
`db.<collection>.drop()`	Completely removes the specified collection `<collection>`

Other Ways to Use MongoDB

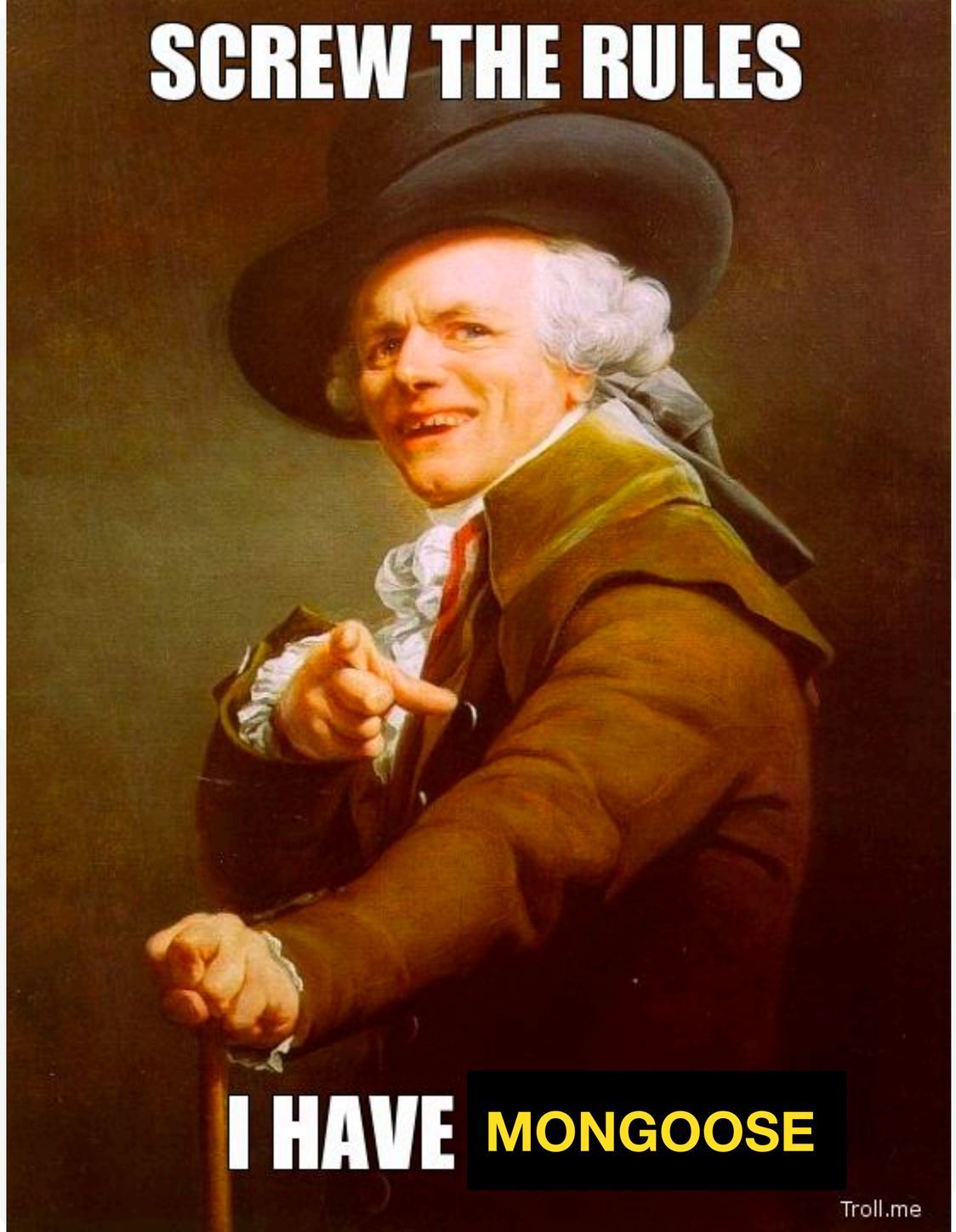
- MongoDB Atlas - Lets you create a remote database instead of a local one (useful for Windows users)
- mLab - What you'll use to create a "production" database when deploying to Heroku
- MongoDB Compass - A handy graphical user interface (GUI) for accessing database collections / documents

Mongoose

- Dependency: `npm install mongoose`
- Our **middleman** between our app and MongoDB
- Has wrapper functions for MongoDB API
- Lets us do:
    - Schema creation
    - Schema validation
    - CRUD operations
    - And more!

Schema Validation

- We wrote our own, but now we'll hand it off to Mongoose
- Every database action that changes data goes through validation
    - Actions return **Promises**
- All based on the rules in the **schema**
- Easy plug-and-play!



Data Modeling

- Essentially what we've been doing with the addition of schemas
- The process of structuring your data and the relationships between data
- Mongoose creates a **data model** for use in our JavaScript code
    - Schema dictates the model!
    - CRUD operations are the actions a model can do for its records
- Next class, we'll make a wrapper around the Mongoose data model
    - Gives us more power + consistency
    - Makes it easier to change things!

Vocab Review

database

database

An organized set of data stored long-term

in an accessible location.

data model

data model

An abstract set of rules or guidelines that categorizes data in an application. Data models define what data should look like, how data relates to one another, and how to access data.

CRUD

CRUD

Create, Read, Update and Delete are the four basic functions for storing and accessing stored data. We use this acronym to keep in mind how we most commonly interact with a database.

schema

schema

A schema is a collection of rules that describes a type of data within our larger data model. Schemas typically define what properties a piece of data should have.

sanitize

sanitize

The process of “cleaning” data to remove any illegal characters, to standardize naming conventions, and to enforce type upon data values.

Structured Query Language (SQL)

Structured Query Language (SQL)

A standard language for interacting with relational databases. SQL relies on “data queries” in the format of “SELECT __ FROM __ WHERE”.

Non SQL (NoSQL)

Non SQL (NoSQL)

Non SQL refers to databases that are non-relational (do not enforce a relation table) and thus do not need to use the SQL language. NoSQL databases can choose what language to use, and they expose custom APIs for communicating with the database within an application.

MongoDB

MongoDB

A cross-platform document-based database that is non-relational. MongoDB uses JSON-like documents that adhere to a certain schema, and arranges these documents into collections.

Mongoose

Mongoose

A package / library for using MongoDB in a Node.js application. Mongoose provides a large suite of features, such as schema validation, API connection to a MongoDB database, and translation between JavaScript code and MongoDB code.

document

document

In MongoDB, each piece of data is stored as a binary json (`.bson`) file on the database server (which can be your local computer if you're running MongoDB locally!). Because each piece of data is stored as a file, data entries are referred to as "documents".

record

record

A more generic term for a data entry in a database. For MongoDB, documents and records are synonymous.

Object Relational Mapping (ORM)

Object Relational Mapping (ORM)

A programming technique of converting data between two incompatible systems.

For example, Mongoose converts JavaScript data into terms MongoDB can understand, and vice-versa.

Lab 03 Overview

Code Challenge 03 Overview

Your Next Week

Saturday March 21

9 AM
— DUE Class 02 Code Challenge
— DUE Class 02 Lab
— DUE Class 03 Reading
— Class 03

MIDNIGHT
— DUE Class 03 Learning Journal

Sunday March 22

MIDNIGHT
— DUE Career: Professional Etiquette
— DUE Class 02 - 03 Feedback

Monday March 23

Tuesday March 24

6:30 PM
— DUE Class 03 Code Challenge
— DUE Class 03 Lab
— DUE Class 04 Reading
— Class 04A

Wednesday March 25

6:30 PM
— Class 04B

MIDNIGHT
— DUE Class 04 Learning Journal

Thursday March 26

6:30 PM
— Lab 04 & Code Challenge 04 Co-Working

Friday March 27

Saturday March 28

9 AM
— DUE ALL PRE-WORK
— DUE Class 04 Code Challenge
— DUE Class 04 Lab
— DUE Class 05 Reading
— Class 05

MIDNIGHT
— DUE Class 04 Learning Journal