Full Stack JavaScript

Joaquin Guardado

Published with GitBook

Table of Contents

Full Stack JavaScript

Welcome, this book is for you if you are just getting started on your journey to become a Web Developer. You will learn the basics necessary to understand and deploy a Web Application using node, express, angular and MongoDB (MEAN).

Something worth mentioning is that before you read this book you should be familiar with the following.

- The command line and common commands
- Github and Git
- Chrome Dev Tools
- A text editor
- JavaScript
- jQuery
- OS X or Linux
- HTML and CSS

There are some free options to learn the basics of Web Development. Head on over to CodeSchool and checkout their free lessons.

Good luck on your journey and I hope this book serves you well.

*Note: This book is a work on progress and get's updated at least once a week.

Pull requests and suggestions are appreciated.

GitBook URL

About Me

I'm a Full Stack JavaScript bootcamp gradute. This book is my contribution to the open source community. My goal is to create a guide to help you get started developing with the MEAN stack and help you on your way to become an awesome developer.

Note:

My development enviroment is in Mac OS X and I highly recommend it. If you must develop in a different environment Linux is the next best thing. Please make sure to have access to either before starting to read this book.

~ JQN ##

Basics

Command Line

It is very important to get familiar with using the command line. The following is a list of the basic commands you will need while creating a web app.

```
cd - traverse directories
pwd - print working (current) directory
mkdir - creates an empty directory
touch - creates new black file
rm - deletes a file
rm -rf - delete a directory and all of its contents. Be careful this command doesn't send
ls - list files in current directory
mv - moves or renames a file
cp - copies a file
open - opens a file with the default program
Shortcuts

ctrl + c - kill current process
```

Environment Setup

Get started with node, install the following tools:

Latest version of Ruby (for Sass, and other tools) Node.js, PostgreSQL, MongoDB, Redis,

Editors: Atom.io or Sublime Text.

Mac OS:

- Homebrew http://brew.sh Note: the instructions are at the end of the web page.
- rbenv, ruby-build, ruby 2.1.0 and the sass gem
  - `brew doctor`
  - `brew update`
  - `brew install rbenv ruby-build rbenv-gem-rehash`
  - `echo &#39;export PATH=&quot;$HOME/.rbenv/bin:$PATH&quot;&#39; &gt;&gt; ~/.bash_profile`
  - `echo &#39;eval &quot;$(rbenv init -)&quot;&#39; &gt;&gt; ~/.bash_profile`
  - `rbenv install 2.1.2`
  - `rbenv global 2.1.2`
  - `gem install sass`
  - Don't use sudo to install ruby or gems
  - If you get a permissions error when installing sass, somehow system ruby is still active. Try restarting your terminal, or if it persists, check for the items above in your .bash_profile
- Node.js
  - `brew install nvm`
  - `nvm install 0.10`
  - `nvm alias default 0.10`
  - add `source $(brew --prefix nvm)/nvm.sh` to your .bash_profile or .zshrc
  - Reference the NVM README if you get stuck
- MongoDB
  - `brew install mongodb`
  - You may not want it to start at login, it's pretty easy to just run mongod when you need it.
  - Skip the instructions after the installation and create the db folder in your the root directory of you project. Then open a new terminal tab and run `mongod --dbpath ~/data/db` to start MongoDB.

    - After that open a second terminal tab and start MongoDB with `mongod`
- Heroku Toolbelt
    - `brew install heroku-toolbelt`
- Git
    - `brew install git`
- Pick a programmer's editor:
    - Atom.io is my personal favorite but a lot of developers choose Sublime Text 3:
        - http://www.sublimetext.com/3
        - Package Control: https://sublime.wbond.net/installation

Optional Installs until needed.

- Redis
    - brew install redis
    - same as above, you don't need it to start at login
- PostgreSQL
    - You can follow this post once you are ready.

Ubuntu:

- No need for homebrew you already have a perfectly good package management system.
    - In your terminal preferences make sure that "Run Command as a login shell is an enabled profile preferences" check these two screenshots: http://cl.ly/image/220M3f093v2M http://cl.ly/image/3i2O0y0A3e04
    - rbenv, ruby-build, and ruby: https://www.digitalocean.com/community/articles/how-to-install-ruby-on-rails-on-ubuntu-12-04-lts-with-rbenv--2
    - NOTE you DO NOT have to buy a digital ocean server, this is instructions for how to install LOCALLY. Ignore the create a server droplet step.
    - NOTE replace 1.9.3 with the latest version of ruby: 2.1.0
- gem install sass // DO NOT use sudo to install gems
- node.js: https://github.com/joyent/node/wiki/Installing-Node.js-via-package-manager
- PostgreSQL
    - Follow this blog post: https://www.codefellows.org/blogs/how-to-install-postgresql
        - Sublime Text 3
    - http://docs.sublimetext.info/en/latest/getting_started/install.html
- MongoDB - https://www.digitalocean.com/community/articles/how-to-install-mongodb-on-ubuntu-12-04

    - NOTE you DO NOT have to buy a digital ocean server, these are instructions to install LOCALLY. Ignore the create a server droplet step.
    - Redis - https://library.linode.com/databases/redis/ubuntu-12.04-precise-pangolin same as above
- Heroku Toolbelt - sudo apt-get install heroku-toolbelt

Arch Linux and Manjaro

To set up your local dev follow this tutorial.

Here is another resource for installing ruby on rails:

DigitalOcean how to install Ruby on Rails.

Git

To install git make sure to have homebrew installed. Run `brew install git` in the Terminal and you are set.

Useful Git Commands

`git ls-remote` | list remote branches

`git config --global --edit` | edit conf file

`git commit --amend --reset-author` | fix indentity used.

If you ever run in trouble like I do the following will help you out.

`rm -rf directory-name` | Removes a directory with contents

`rm -rf .git` | Removes git from directory. Make sure that you absolutely need to this, it will delete everything from branches to commits.

`git reset --hard HEAD` | This will bring you back to your last commit and will ignore any new changes made after.

`git branch -d branch-name&#39;` | Deletes a branch.

Atom

One of the coolest features of atom.io is to allow you to work in the command line within the same interface. This package comes bundled with keybindings that will help you be faster and more productive.

To install search for term2 in packages within atom or from the terminal run $apm install term2. Once is done installing try this keybinding to open it up.

Description	Keybinding
term2	ctrl-alt-arrow-key

Here is another cool feature worth mentioning, markdown preview is native to atom and will let you preview your README.md from a separate pane within atom.io.

Description	Keybinding
Markdown	ctrl + shift + m

Some cool and useful keyboard shortcuts

Description	Shortcut
Toggle command palette	shift + ⌘ + p
Toggle Sidebar	⌘ + /
Multiple Cursors	ctrl + shift + up
Move a line of code up or down	ctrl + ⌘ + up or down

Update:

To fix the pty.js install error do the following instead of running `$ apm install`

```
git clone https://github.com/svenax/atom-term2.git
cd atom-term2
git checkout update-pty.js
apm install
sudo apm link .
```

Restart Atom and term2 should be available in your packages list.

To resolve the newest problem when pressing k on the keyboard

```
cd ~/.atom/packages/term2/node_modules
rm -rf atom-term.js
git clone https://github.com/mmckegg/atom-term.js.git atom-term.js
cd atom-term.js
git checkout patch-1
```

… then restart Atom.

MongoDB

Install MongoDB with homebrew

Update homebrew's package database

```
brew update
```

Install MongoDB

```
brew install mongodb
```

To have launchd start mongodb at login:

```
ln -sfv /usr/local/opt/mongodb/*.plist ~/Library/LaunchAgents
```

Then load mongodb:

```
launchctl load ~/Library/LaunchAgents/homebrew.mxcl.mongodb.plist
```

If you don't need launchtl just run:

```
mongod --config /usr/local/etc/mongod.conf
```

Create the data directory.

```
mkdir -p /data/db
```

In your project root directory

Run `mongo`

Select database `db`

Show databases `show dbs`

Switch to your database `use mydb`

Confirm `db`

Show collections `show collections`

Display documents `db.testData.find()`

If you want to run MongoDB on demand skip the autostart on login step. Once you create your project folder `mkdir` an empty db folder and open two tabs in your terminal.

Run:

```
mongod --dbpath ~/data/db
```

and in the other tab run:

```
mongod
```

`brew info mongodb` for reinstall instructions

Grunt

Once your local dev environment is set-up, it's a good idea to get started with Grunt. This is an extremely powerful JavaScript task runner, a build tool and and automation tool.

Some typical Grunt taks:

- Concatenation.
- Minification.
- Pre-processing Sass and Coffeescript.
- Image Optimization.
- Running tests.
- Starting a development Server.

Installation:

Global: `npm -g install grunt-cli`

Dev: `npm install grunt --save-dev`

Create a package.json file in your root directory with `npm init`

```
//package.json
{
  "name": "my-project",
  "version": "0.0.1",
  "devDependencies": {
    "grunt": "~0.4.4"
  }
}
```

Create a Gruntfile.js

```
module.exports = function(grunt) {

  // Project configuration.
  grunt.initConfig({
    simplemocha: {
      options: {
        timeout: 3000,
        ignoreLeaks: false,
        reporter: 'tap'
      },

      all: { src: ['test/**/*.js'] }
    },
    jshint: {
      all: ['Gruntfile.js','test/**/*.js']
    }
  });

  // Simplemocha test
  grunt.loadNpmTasks('grunt-simple-mocha');

  // JSHint
  grunt.loadNpmTasks('grunt-contrib-jshint');

  // Default task.
  grunt.registerTask('default', 'simplemocha');

};
```

`npm install` to install dependencies.

Here is an example of Grunt compiling SASS.

grunt-sass-boilerplate

Angular Project Set-up

Folder structure: app/server.js

Create a package.json file `npm init`

Install express as a dependency `npm install express --save`

Install nodemon as a dev dependency `npm install nodemon --save-dev`

Create server.js

At this point if you are only working on the html and css is best to use nodemon to reload your page automatically every time a change is made. Run with `$ nodemon` in a separate shell tab.

Sass

Server Auth

JSON Authentication with Passport and Basic HTTP

Setup

The first Step is to create a node server with a JSON api

package.json

```
{
  "name" : "auth-json-api",
  "description" : "auth json api",
  "version" : "0.0.1",
  "dependencies" : {
    "express" : "^4.x",
    "passport" : "^0.2",
    "passport-http" : "^0.2"
    "mongoose" : "^3.8",
    "bcrypt" : "latest",
    "jwt-simple" : "^0.2",
    "moment" : "^2.7"
  }
}
```

$ npm install dependencies from the root folder.

Generate a self-signed ssl certificate and key and put them in a folder named config.

server.js

models/user.js

lib/authentication/passportBasic.js

routes/userRoutes.js

lib/authentication/jwtAuth.js

Heroku Deployment

This guide will go over deploying your site to Heroku and setting up a MongoLab database for your site. We'll be using the commad line to run our commands.

Installation

Start by installing Heroku Toolbelt:

`brew install heroku-toolbelt` or `sudo apt-get install heroku-toolbelt`

Sign up for an account on Heroku.com.

Login

Use `heroku login` to login into heroku.

In case you are alredy logged in, use `heroku auth:whoami` b to reveal who you are logged in as.

Create a heroku app

```
heroku create &lt;appname&gt;
```

```
git push heroku master
```

The heroku deployment is done at this point. Open your favorite browser and point it to the heroku url.

My favorite way to open my new heroku app is using `` `heroku open ``

Set-up database with MongoLab

Back on the command line:

`heroku addons:add mongolab` Will ask for verification.

`heroku config` Displays mongo.uri. We have to st this to the mongoURI

```
heroku config:set MONGO_URL=&lt;mongo uri&gt;
```

```
heroku config:set STATIC_DIR=&quot;/dist&quot;
```

Sign up for a free-tier sandbox account at mongolab.com.

Add a remote connection, set name to your databse name and add your MongoURI.

Function Expressions

This function builds in memory immediately when the program loads.

```
function diffOfSquares (a, b) {
    return a*a - b*b;
    }
```

Function expressions build only when the program reaches their line of code.

```
var diff = function (a, b) {
 return a*a - b*b;
};

diff( 9, 5 ); // This is how we call the function
```

A variable that holds a function can be passed into other functions. Function expressions can give flexibility in choosing which functionality to build.

```
var greeting;
var newCustomer =  false;

if (newCustomer) {
  greeting = function() {
    alert("Welcome first timer!) {
    };
  } else {
    alert("Welcome back beloved customer");
  };
}
closeTerminal (greeting);
function closeTerminal(message){
  message();
}
```

Objects

Objects are containers of related information. Multiple pieces of data, called properties, are grouped within an Object.

Everyday things can be represented with JS Objects, for instance we could represent a book.

Book Object
Properties
title
author
publisher
illustrator
country

Properties

Every property is an important bit of data associated with a book. An Object also contains multiple bits of info, this is often called a "composite value".

Values

Just like everyday Objects, properites can point to specific ammonts or qualities. This means that properties can have assigned values.

Book Object	
Properties	Values
title	Twenty Thousand Leagues Under the Sea
author	Jules Verne
publisher	Pierre-Jules Hetzel
illustrator	Alphonse de Neuville
country	France

Creating Objects

There are several ways to create Objects in JS. One way is the Object literal. This is also the most often used.

```
var myFavoriteBook = {};
```

A set of curly brackets means to make a new Object. In this case it's an empty object with no properties.

To add a property create a name for the property using a string and then assign a value to it using a :

```
var myFavoriteBook = { title: "Twenty Thousand Leagues Under the Sea",
                       author: "Jules Verne",
                       publisher: "Pierre-Jules Hetzel",
                       illustrator: "Alphonse de Neuville",
                       country: "France"
                       };
```

Properties can have arrays of values and also accept variables.

```
var chapters = ["A Runaway Reef", "The Pros and Cons", "As Master Wishes",
                "Ned Land", "At Random!", "At Full Steam",
                "A Whale of Unknown Species", "Mobilis in Mobili"];

var myFavoriteBook = { title: "Twenty Thousand Leagues Under the Sea",
                       author: "Jules Verne",
                       publisher: "Pierre-Jules Hetzel",
                       illustrator: "Alphonse de Neuville",
                       country: "France",
                       contents: chapters
                       };
```

- Notice how we have a reference to the chapters inside the book Object.

Referencing Properties

One way to peek at a property inside of an Object is to use dot notation.

Let's peek inside our book Object.

```
var chapters = ["A Runaway Reef", "The Pros and Cons", "As Master Wishes",
                "Ned Land", "At Random!", "At Full Steam",
                "A Whale of Unknown Species", "Mobilis in Mobili"];

var myFavoriteBook = { title: "Twenty Thousand Leagues Under the Sea",
                       author: "Jules Verne",
                       publisher: "Pierre-Jules Hetzel",
                       illustrator: "Alphonse de Neuville",
                       country: "France",
                       contents: chapters
                       };
```

```
myFavoriteBook.author; // -&gt; &quot;Jules Verne&quot;
```

This example returns the name of the author.

```
myFavoriteBook.contents;
// -> ["A Runaway Reef", "The Pros and Cons", "As Master Wishes",
                "Ned Land", "At Random!", "At Full Steam",
                "A Whale of Unknown Species", "Mobilis in Mobili"]
```

The previous example returns the array inside chapters.

Changing Property Values

Dot notation or the dot operator also allows modification of properties, even when using methods.

Here is our book Object again.

```
var chapters = ["A Runaway Reef", "The Pros and Cons", "As Master Wishes",
                "Ned Land", "At Random!", "At Full Steam",
                "A Whale of Unknown Species", "Mobilis in Mobili"];

var myFavoriteBook = { title: "Twenty Thousand Leagues Under the Sea",
                       author: "Jules Verne",
                       publisher: "Pierre-Jules Hetzel",
                       illustrator: "Alphonse de Neuville",
                       country: "France",
                       contents: chapters
                       };
```

To change the name of the author we could use:

```
myFavoriteBook.author = &quot;Bob&quot;;
```

This would change the name of the author from Jules Verne to Bob.

With dot notation we can also modify the contents of a property.

```
myFavoriteBook.contents.push(&quot;The Tantrums of Ned Land&quot;);
```

Notice the use of the push method. This allows us to add or in this case push the missing chapters in our contents property.

Adding Properties

Even if an Object is already created, properties can continue to be added.

```
var chapters = ["A Runaway Reef", "The Pros and Cons", "As Master Wishes",
                "Ned Land", "At Random!", "At Full Steam",
                "A Whale of Unknown Species", "Mobilis in Mobili",
                "The Tantrums of Ned Land"
                ];

var myFavoriteBook = { title: "Twenty Thousand Leagues Under the Sea",
                       author: "Jules Verne",
                       publisher: "Pierre-Jules Hetzel",
                       illustrator: "Alphonse de Neuville",
                       country: "France",
                       contents: chapters,
                       publicationDate: 1970
                       };
```

```
myFavoriteBook.publicationDate = 1970;
```

The book Object first attempts to look for a publicationDate property. If it doesn't find one, it creates one.

Accessing and creating properties with the bracket method.

This method is similar to accessing array indices. We must pass in a string in order to reference a property.

```
myFavoriteBook[&quot;author&quot;];
```

The bracket method comes in handy if we want to add a property name with spaces and characters.

```
myFavoriteBook[&quot;Original Title&quot;] = &quot;Vingt mille lieues sous les mers&quot;;
```

To use console.log with this type of method:

```
console.log(myFavoriteBook[&quot;Original Title&quot;]);
```

Brackets also enable dynamic property access. Since they take expressions, we can avoid hard-coding every property access.

Lets create a function to add more chapters to our Object and to turn chapters into idividual Objects.

```
var chapters = ["A Runaway Reef", "The Pros and Cons", "As Master Wishes",
                "Ned Land", "At Random!", "At Full Steam",
                "A Whale of Unknown Species", "Mobilis in Mobili",
                "The Tantrums of Ned Land"
                ];

var myFavoriteBook = { title: "Twenty Thousand Leagues Under the Sea",
                       author: "Jules Verne",
                       publisher: "Pierre-Jules Hetzel",
                       illustrator: "Alphonse de Neuville",
                       country: "France",
                       publicationDate: 1970,
                       "# of chapters": 0,
                       chapter1: {
                       };

function addBook ( book chapter title ){
    chapter
}
```

Referencing Objects within Objects

To get to the deeper properties of an object within another object we can use dot extension or subsequent bracket notation.

```
var myBookBox = { height: 8, length: 10, width: 12,
              book1: {title: "Strange Case of Dr Jekyll and Mr Hyde", author: "Robert Lou
}
```

Dot notation:

```
console.log(myBookBox.book1.title);
```

This logs out the Strange Case of Dr Jekyll and Mr Hyde.

Wit bracket notation: `console.log( myBookBox[&quot;book1&quot;][&quot;author&quot;] );`

This logs out Robert Louis Stevenson.

Data Structures

Arrays

Linked List

Arrays are not always the optimal solution for organizing data. Whenever the operations performed in an array are too slow for practical use, it is usually best to use a linked list. They can be used in almost any case where a one-dimensional array is used, except when random access to the list is needed. An array is the best choice in this situation.

A linked list is a collection of nodes or objects, linked to a successor node in the list using and object reference. This references are called links.

Linked list elements unlike array elements are not referenced by their position, rather by their relationship to the other elements of the linked list. For instance Robin follows Batman.

The beginning of a linked list is marked with a head node and the end with a null node.

Header -> Alfred -> Robin -> Batman -> Null

Inserting a new node in a linked list requires pointing the link before the inserted node to the new node, and the new node's link is set to the node the previous node was pointing to before insertion.

Removing nodes the link of the node before the removed node is redirected to point to the node tehremoved kinothe removed node was pointing to, while also pointing the removed node to null.

Implementing a Linked List

Designing a linked list requires the creation of two classes. Number one is a Node class for adding nodes to the list and number two a LinkedList class. This last class will provide functions for inserting, removing and deleting nodes plus displaying the list, and basic housekeeping functions.

Node Class

The Node class contains two properties, value, to store the node's data, and nexNode to store a link to the next node in the linked list.

The following is a constructor function that sets the values for the two properties.

```
function Node(data) {
  this.data = data;
  this.nextNode = null;
}
```

Linked List Class

Here we add the functionality for a linked list. We have to include functions that will be able to insert, remove or delete nodes, as well as finding data values in the list.

Here is the constructor function necessary to create a new linked list.

```
function LinkedList() {
  this.head = new Node("head"); //points null to the first element inserted
  this.find = find;
  this.insert = insert;
  this.remove = remove;
  this.display = display;
}
```

Manipulating a Linked List

Node Insertion

First we need a helper function that searches through the linked list looking for specified data, when this data is found, the function returns the node storing the data. This function is also a good example of moving through a linked list.

The function is built as follows.

- Create a new node and assign it as the head node.
- Loop through the linked list, from one node to the next when the current node's property is not equal to the data we are looking for.
- When this data is found return the node storing the data.
- If the data isn't found return null.

```
function find(item) {
  var currentNode = this.head;
  while (current.element != item) {
    currentNode = currentNode.next;
    }
    return currentNode;
  }
```

The insert function.

Singly Linked List

```
function Node(element) {
  this.element = element;
  this.next = null;
}

function LList() {
  this.head = new Node("head");
  this.find = find;
  this.insert = insert;
  this.display = display;
}

function find(item) {
var currNode = this.head;
while (currNode.element != item) {
      currNode = currNode.next;
   }
return currNode;
}

function insert(newElement, item) {
  var newNode = new Node(newElement);
  var current = this.find(item);
  newNode.next = current.next;
  current.next = newNode;
}

function display() {
var currNode = this.head;
while (currNode.next !== null) {
      console.log(currNode.next.element);
      currNode = currNode.next;
   }
}


var cities = new LList();
cities.insert("Seattle", "head");
cities.insert("Portland", "Seattle");
cities.insert("Boulder", "Portland");
cities.display();
```

Stack

Binary Trees

A binary tree is a tree structure such that each node, apart from the final nodes, have exactly two children. This type of non-linear data structure is used to store data in a hierarchichal manner, for instance lists of data. Binary trees can be searched very quickly (as opposed to a linked list) and data can be inserted or deleted very quickly (as opposed to an array).

To specify the element of the tree we have to give it two indices - the level or how far down the tree and the node number at that lever - how far to the right the node is.

Each value is associated with a value that takes n bytes to store.

Anatomy

Trees are made up of a set of nodes connected by edges. A good example would be an organizational chart.

The top of node of a tree is called root node. If a node is connected to other nodes below it, the preceding node is called the parent node, and the nodes following it are called child nodes.

A node can have zero, one, or more child nodes connected to it. A node without any child nodes is called a leaf node.

In the case of binary trees the number of child nodes is restricted to no more than two.

It is possible to travel from one node to other nodes that are not directly connected. The series of edges needed to get from one node to the other are called a path. Visiting all the nodes in a tree in some particular order is known as a tree traversal.

A tree can be broken down into two levels. The root node is at level 0, its children are at level 1, those nodes' children are at level 2, and so on. We can define the depth of a tree as the number of layers in the tree.

Each node in a tree has a value associated with it. This value is referred as the key value.

Binary Tree Efficiency

Because of their limiting nature, no more than two children, it is possible to write efficient programs for inserting, searching, and deleting data.

Tree Lingo

The child nodes of a parent node are referred to as the left node and the right node.

Binary Search Trees

A binary search tree or a BST is a binary tree in which data with lesser values are stored in left nodes and data with greater values is stored in right nodes. This characteristics provide for very efficient searches and holds for both numeric data and non-numeric data.

Binary Search Tree Implementation

A binary search tree is made up of nodes, therefore we need to start with creating a **Node object.**

```
function Node (data, left, right) {
  this.data = data;
  this.left = left;
  this.right = right;
  this.show = show;
}

  function show() {
  return this.data;
}
```

The Node object stores data and links to other nodes (left & right).

The show( ) function displays the data stored in a node.

Now we can build a **class** to represent a BST. The class has to consist of one data member or a Node object that represents the root node. The constructor for the class sets the root node to null, creating an empty node.

Then we need an **insert** function to add new nodes to the tree. This function will initially create a Node object, passing in the data the node will store. Secondly we check the BST for a root node. If a root node doesn't exist, the BST is new and this node is the root node, this completes the function definition.

If the node being inserted is not the root node, then we have to traverse the BST to find the proper insertion point. The function uses a Node object that is assigned as the current node as the function moves from level to level in the BST. This function also has to position itself inside the BST at the root node.

Once inside the BST, the next step is to determine where to put the node. This is performed inside a loop that breaks once the correct insertion point is determined.

The algorithm for determining the correct insertion point for a node is as follows.

1. Set the root node to be the current node.
2. If the data value in the inserted node is less than the data value in the current node, set the new current node to be the left child of the current node. If the data value in the inserted node is greater than the data value in the current node, skip to step 4.
3. If the value of the left child of the current node is null, insert the new node here and exit the loop. Otherwise, skip to the next iteration of the loop.
4. Set the current node to be the right child of the current node.
5. If the value of the right child of the current node is null, insert the new node here and exit the loop. Otherwise, skip to the next iteration of the loop.

With this algorithm complete, we're ready to implement this part of the BST class.

BST class

```
function BST() {
  this.root = null;
  this.insert = insert;
  this.inOrder = inOrder;
}

function insert(data) {
  var n = new Node(data, null, null);
  if (this.root === null) {
    this.root = n;
} else {
  var current = this.root;
  var parent;
  while (true) {
    parent = current;
    if (data < current.data) {
      current = current.left;
      if (current === null) {
        parent.left = n;
        break;
      }
    } else {
      current = current.right;
      if (current === null) {
        parent.right = n;
        break;
        }
      }
    }
  }
```

Traversing a Binary Search Tree

Our previous BST class can only insert nodes into the tree. In order to be able to display the data in different orders, we need to able to traverse the tree.

There are three traversal functions used with BSTs: inorder, preorder, postorder.

inorder: visits all of the nodes of a BST in ascending order of the node key values.

preorder: visits the root node first, followed by the nodes in the subtrees under the left child of the root node, followed by the nodes in the subtrees under the right child of the root node.

postorder: visits all the child nodes of the left subtree up to the root node, and then visits all of the child nodes of the right subtree up to the root node.

Inorder traversal function

```
function inOrder(node) {
  if (node !== null) {
    inOrder(node.left);
    putstr(node.show() + " ");
    inOrder(node.right);
  }
}
```

Test

```
load("bst.js");
var nums = new BST();
nums.insert(23);
nums.insert(45);
nums.insert(16);
nums.insert(37);
nums.insert(3);
nums.insert(99);
nums.insert(22);
nums.inOrder(nums.root);
```

The output would be: Inorder traversal: 3 16 22 2 3 37 45 99

Hash Table

Algorithms

Quick Sort

The Job Search

How to write a Developer CV/Resume that will get you hired

Resources

Books

JavaScript Web Applications - by Alex MacCaw - O'Reilly

Testable JavaScript - by Mark Ethan Trostles - O'Reilly

JS: The Good Parts - by Douglas Crockford

Secrets of the JavaScript Ninja - by Job Resig and Bear Bibeault

JavaScript Enlightenment - by Code Lindley

JavaScript Pocket Reference - by David Flanagan

HTML5 and JavaScript Web Apps - by Wesley Hales

JavaScript: The Definitive Guide, 6th Edition - by David Flanagan

Articles

JavaScript Style Guide

JavaScript Playground

Express JS Getting Started Guide

VowsJS

Videos

O'Reilly Webcast: How to Build a Chatroom in JavaScript in Under an Hour

Node.js: A Jumpstart for Devs

NodeSchool

Coding Skills

Codility

Tutorial Sites

bento.io

Edevate.com

https://www.edx.org/

Blogs

Technical Interviews

http://www.schneems.com/ut-rails/

Links

Useful Links

i-programmer.info

Good interview questions with solutions: http://www.thatjsdude.com/

Single Page Apps

Spontaneous

http://54.69.47.125/

Break App

http://break-app-2.herokuapp.com/

Instapitch

http://instapitch.herokuapp.com/

sextant.io

Tutorius

http://tutorius.herokuapp.com

Useful Keyboard Shortcuts

Technical Challenges

The Square of a number

```
var square = function(x, y) {
  y = x;
  return x * y;
};

console.log(square(8));
```

Exponential

```
//Write a function that calculates x to the power of z

function power(num, exp) {
  var value = num;

  for (var i = 1; i < exp; i = i + 1) {
    value = value * num;
    //Keeps a running total
  }
  return value;
}

var value = power(2, 3);
console.log(value);

// A loop is exactly like an exponent or multipliying a
// number by itself for a certain amount of times.
// a loop executes one of more statements for a certain amount
// of times.
```

Reverse a string

```
var string = "Batman";

var reverse = function (string) {
  var newString = "";
  for (var i = string.length; i >= 0; i--) {
  newString += string.charAt(i);
  }
  return newString;
};

console.log(reverse(string));
```

Multiply Array

```
var numArray = [2, 3];

function multiply(numArray) {
  var sum = 1;
  for(var i = 0; i < numArray.length; i++){
    sum = sum * numArray[i];
  }
  return sum;
}

console.log(multiply(numArray));
```

Date

```
/*Write a function that converts user entered date formatted as M/D/YYYY to a
format required by an API (YYYYMMDD). The parameter "userDate" and the return value are s

var userDate = "12/31/2014";

function formatDate(userDate) {
  var year = userDate.slice(6, 10);
  var theRest = userDate.slice(0, 5);
  return year + theRest.split('/').join('');
   // format from M/D/YYYY to YYYYMMDD
}

formatDate(userDate);

console.log(formatDate(userDate));
```

Eliminate Duplicates

```
var numArray = [2, 3, 6, 4, 8, 10, 11, 13, 1, 2, 2, 3,3,3,3];
//sort numbers in order
numArray.sort(function(a, b) {return a-b;});
//empty array to store results
var result = [];
//loop through array
for (var i = 0; i < numArray.length; i++) {
    //compare results
    if (numArray[i + 1] === numArray[i]) {
       //push duplicates
        result.push(numArray[i]);
    }
}
//log the contents of the array
console.log(numArray);
//log the repeating numbers in array
console.log("these numbers repeat: " + result);

//made possible by http://dreaminginjavascript.wordpress.com/
//Eliminating duplicates
function eliminateDuplicates(arr) {
  var i,
      len=arr.length,
      out=[],
      obj={};

  for (i=0;i
```

Find the remainder of a number

x - (z * y) x - zy = remainder

Frog Jump

Common Errors

`bash_profile permission denied` - my fix was to delete all the contents from .bash_profile. From the home directory open this file with your favorite text editor.

Version Number

Currently in Beta 0.1.2