Lab 06 Exercise Solutions

Exercise 1: Register Users

As well as storing the donations in the server bound objects:

```
server.bind({
  donations: [],
});
```

Try also storing a list of users - in a similar manner to to the donations:

```
server.bind({
  users: [],
  donations: [],
});
```

Using the donations controller as a guide, see if you can populate this array with new users as they are registered. You will need to write a new route for the signup form:

```
  { method: 'POST', path: '/register', config: Accounts.register },
```

and a matching handler:

```
exports.register = {

  handler: function (request, reply) {
    reply.redirect('/home');
  },

};
```

Exercise 2: Current User

Try also to keep track of the current user:

```
server.bind({
  currentUser : {},
  users: [],
  donations: [],
});
```

Adjust your login controller to update this field.

On the report - include an extra column - `donor` - which should list the name of the donor (the user who is currently logged in),

Solution - declare server bound objects + route

index.js

```
server.bind({
  currentUser: {},
  users: {},
  donations: [],
});
```

routes.js

```
  { method: 'POST', path: '/register', config: Accounts.register },
```

- Store users as an Object, rather than an array.
- This object will contain multiple ‘user’ objects, keyed using the email of each new user object.

Solution - preload users

- initUsers an object literal
- It contains 2 name/value pairs
  - Name is an email of a user
  - Value is an object
- server users initialised with initUsers

```
const initUsers = {
  'bart@simpson.com': {
    firstName: 'bart',
    lastName: 'simpson',
    email: 'bart@simpson.com',
    password: 'secret',
  },
  'lisa@simpson.com': {
    firstName: 'lisa',
    lastName: 'simpson',
    email: 'lisa@simpson.com',
    password: 'secret',
  },
};

server.bind({
  currentUser: {},
  users: initUsers,
  donations: [],
});
```

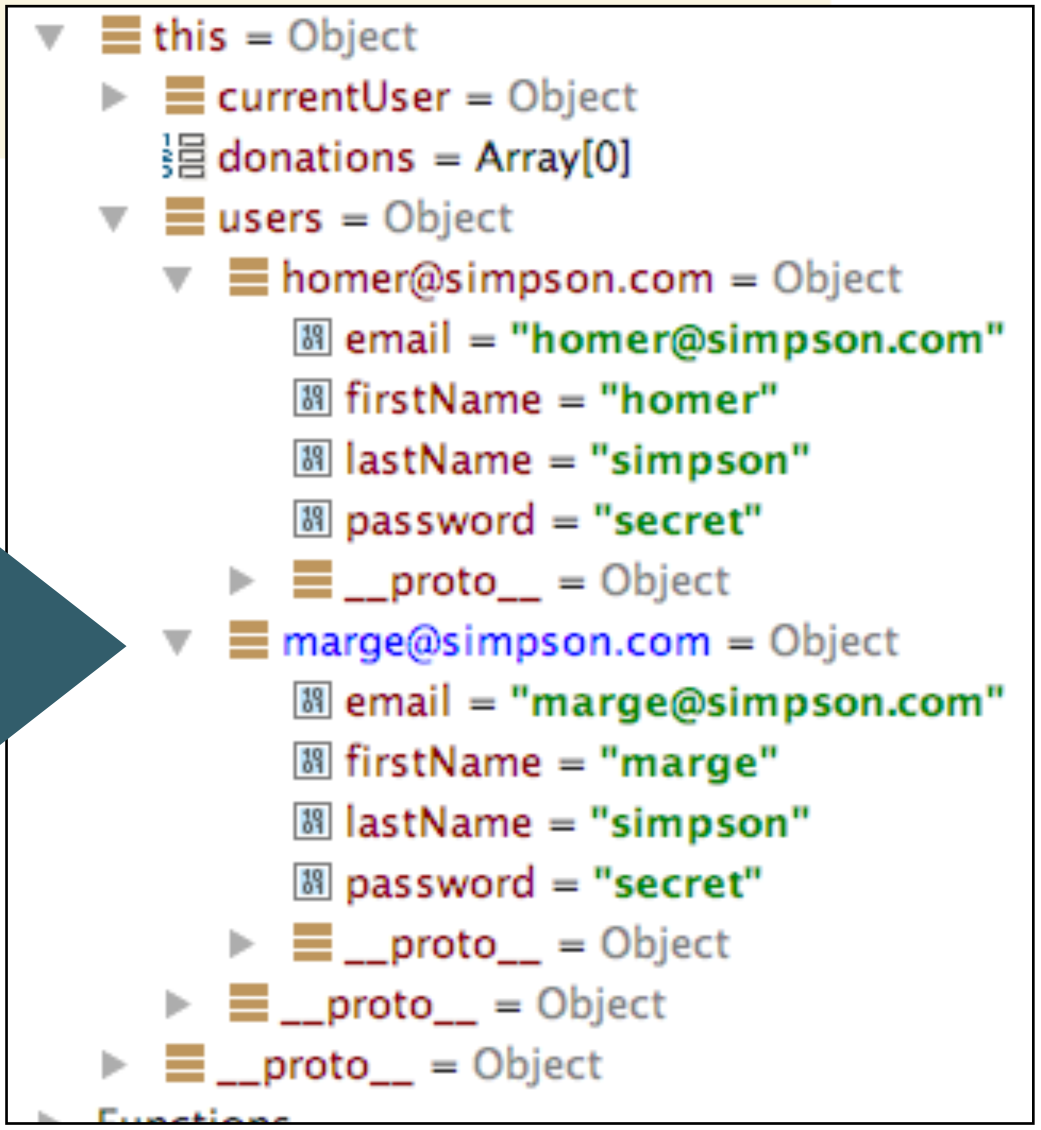
Solution - implement register

```
server.bind({
  currentUser: {},
  users: {},
  donations: [],
});
```

app/controllers/accounts.js

```
exports.register = {

  handler: function (request, reply) {
    const user = request.payload;
    this.users[user.email] = user;
    reply.redirect('/login');
  },

};
```

- ‘users’ defined as a server-bound object.
- Insert new User objects, keyed by the new users email



```
▼ this = Object
  ▶ currentUser = Object
    donations = Array[0]
  ▼ users = Object
    ▼ homer@simpson.com = Object
        email = "homer@simpson.com"
        firstName = "homer"
        lastName = "simpson"
        password = "secret"
      ▶ __proto__ = Object
    ▼ marge@simpson.com = Object
        email = "marge@simpson.com"
        firstName = "marge"
        lastName = "simpson"
        password = "secret"
      ▶ __proto__ = Object
    ▶ __proto__ = Object
  ▶ __proto__ = Object
▶ Functions
```

Solution 3 - implement authenticate, storing current user

```
exports.authenticate = {

  handler: function (request, reply) {
    const user = request.payload;
    if ((user.email in this.users) && (user.password === this.users[user.email].password)) {
      this.currentUser = this.users[user.email];
      reply.redirect('/home');
    } else {
      reply.redirect('/signup');
    }
  },

};
```

- Looking up a user simplified (not need to iterate through an array)
- Reach directly into the users object, using the key (email) field

Solution 4 - have donate record donor (current user)

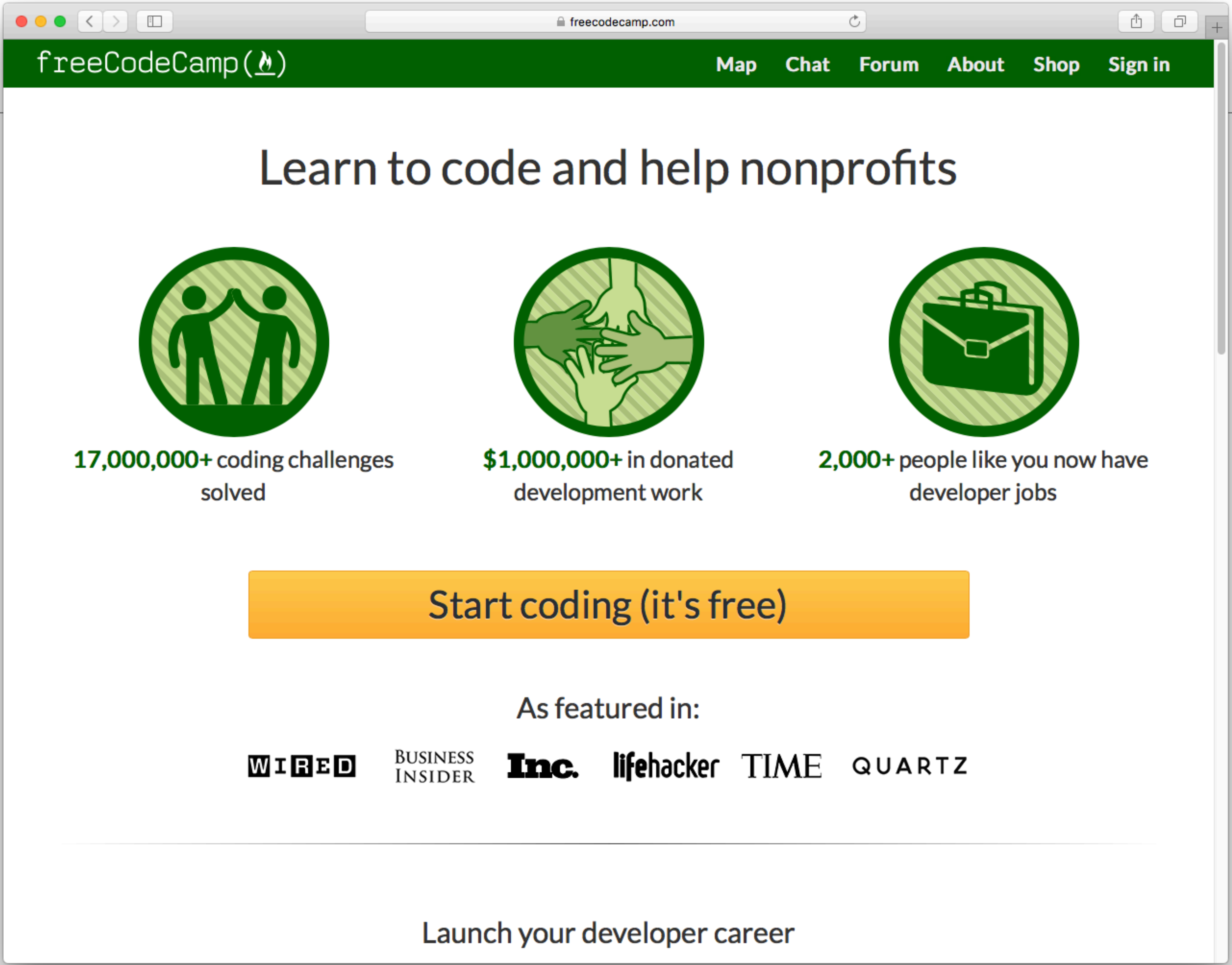
app/controllers/donations.js

```
exports.donate = {

  handler: function (request, reply) {
    let data = request.payload;
    data.donor = this.currentUser;
    this.donations.push(data);
    reply.redirect('/report');
  },

};
```

app/views/partials/donationlist.hbs

```
...
          <tr>
            <th>Amount</th>
            <th>Method donated</th>
            <th>Donor</th>
          </tr>
...
            <tr>
              <td> {{amount}} </td>
              <td> {{method}} </td>
              <td> {{donor.firstName}} {{donor.lastName}} </td>
            </tr>
...
```

JavaScript Skills - FreeCodeCamp



Learn to code and help nonprofits

17,000,000+ coding challenges solved

$1,000,000+ in donated development work

2,000+ people like you now have developer jobs

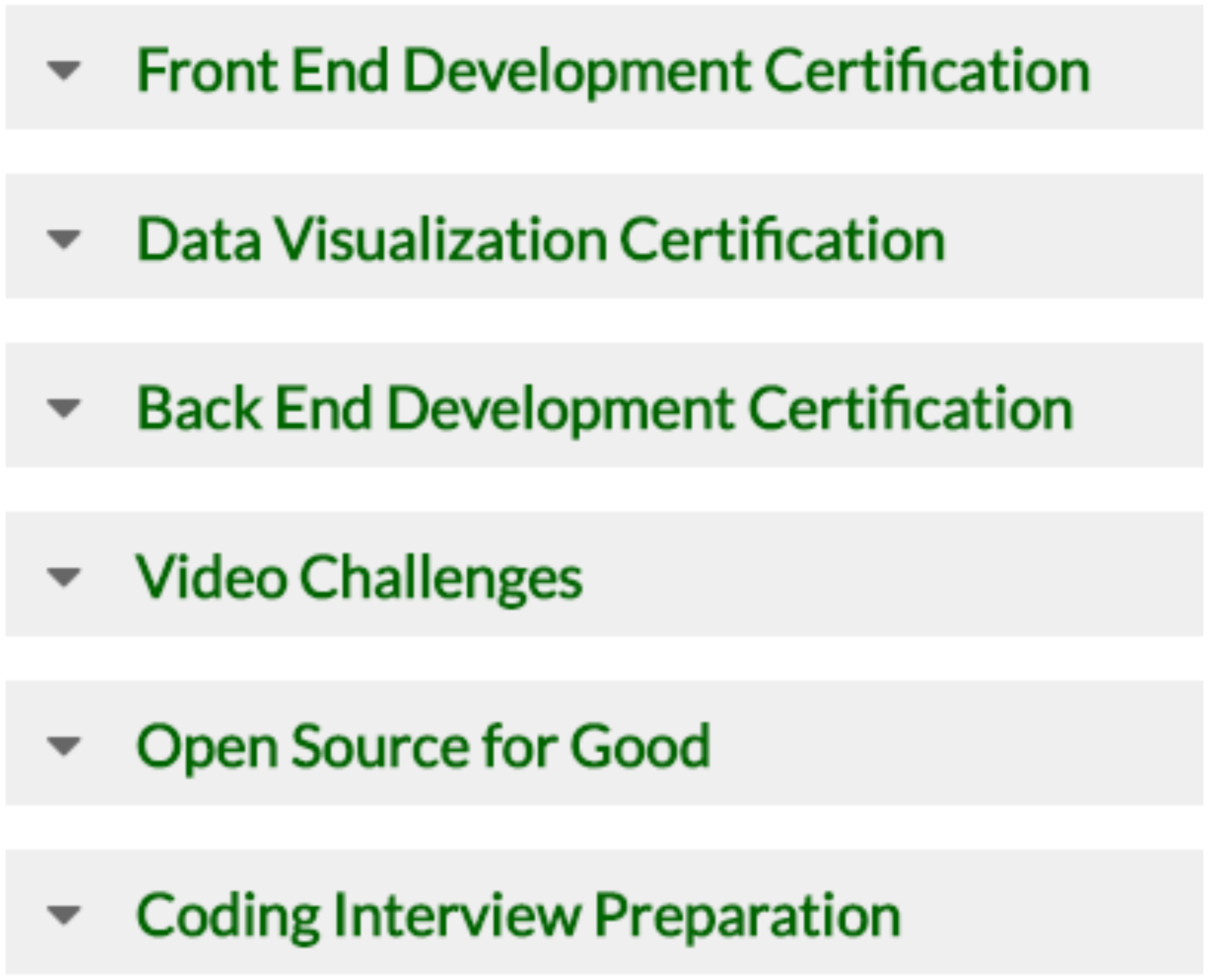
As featured in:

Launch your developer career

JavaScript Programming

- Large proportion of curriculum devoted to javascript skills
- Front End Development contains excellent JavaScript practice problems/solutions

Course Map



Front End Development Certification

Data Visualization Certification

Back End Development Certification

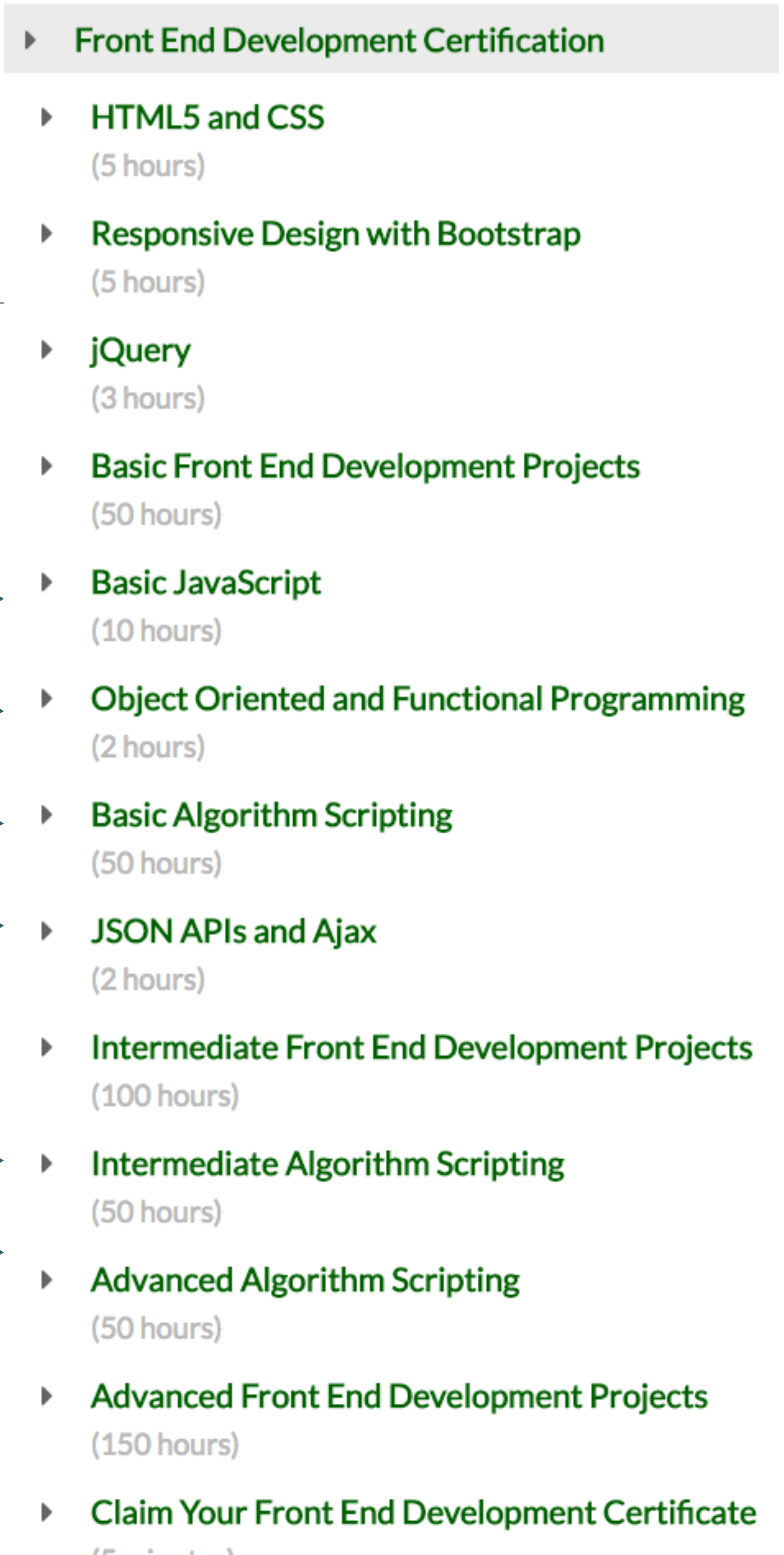
Video Challenges

Open Source for Good

Coding Interview Preparation

Front End Development

- 5 sections - 162 hours of practice
  - Basic Javascript
  - Object Oriented & Functional Programming
  - Basic Algorithm Scripting
  - Intermediate Algorithm Scripting
  - Advanced Algorithm Scripting

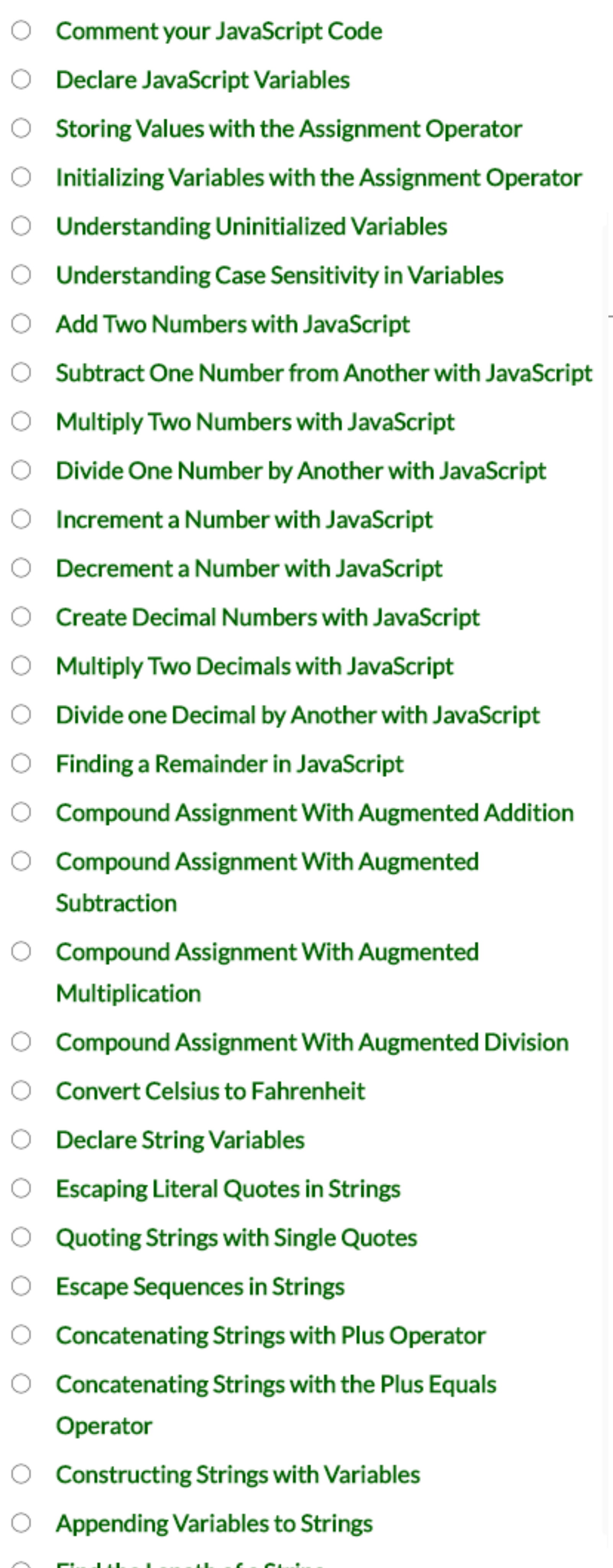


Front End Development Certification

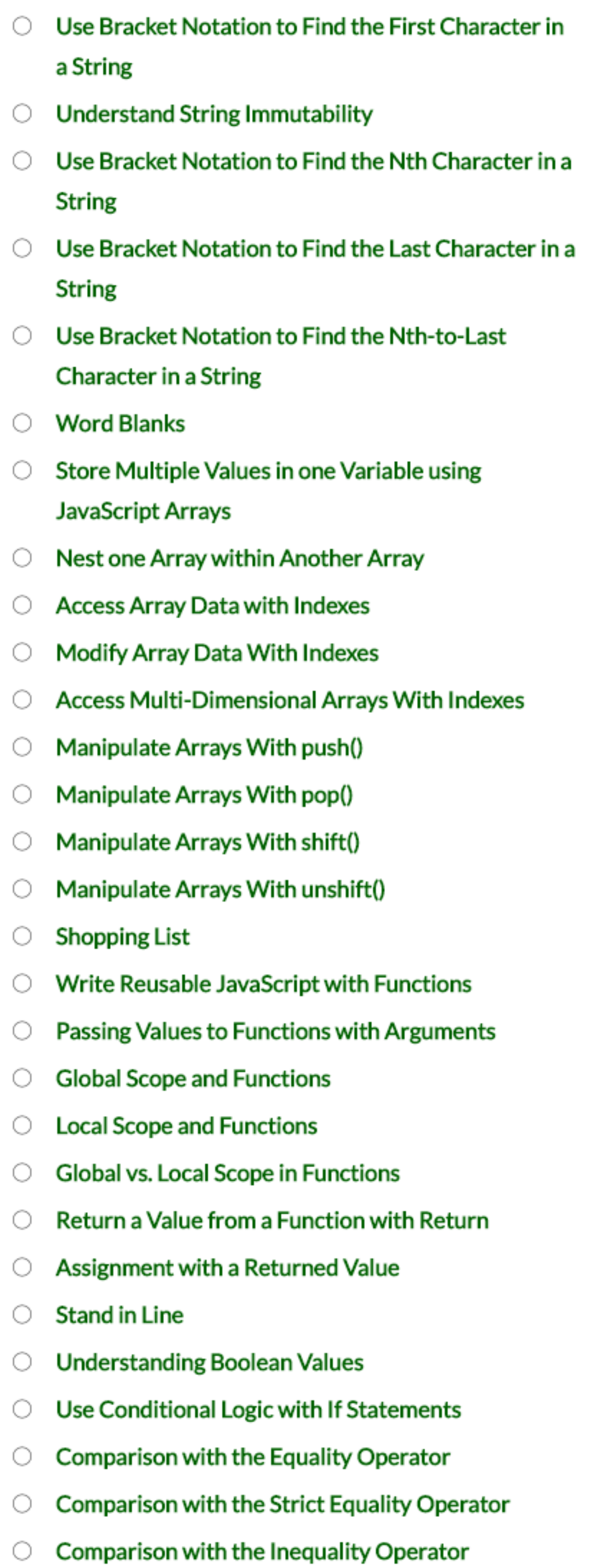
- HTML5 and CSS (5 hours)
- Responsive Design with Bootstrap (5 hours)
- jQuery (3 hours)
- Basic Front End Development Projects (50 hours)
- Basic JavaScript (10 hours)
- Object Oriented and Functional Programming (2 hours)
- Basic Algorithm Scripting (50 hours)
- JSON APIs and Ajax (2 hours)
- Intermediate Front End Development Projects (100 hours)
- Intermediate Algorithm Scripting (50 hours)
- Advanced Algorithm Scripting (50 hours)
- Advanced Front End Development Projects (150 hours)
- Claim Your Front End Development Certificate

Basic Javascript (1)

- 10 hours

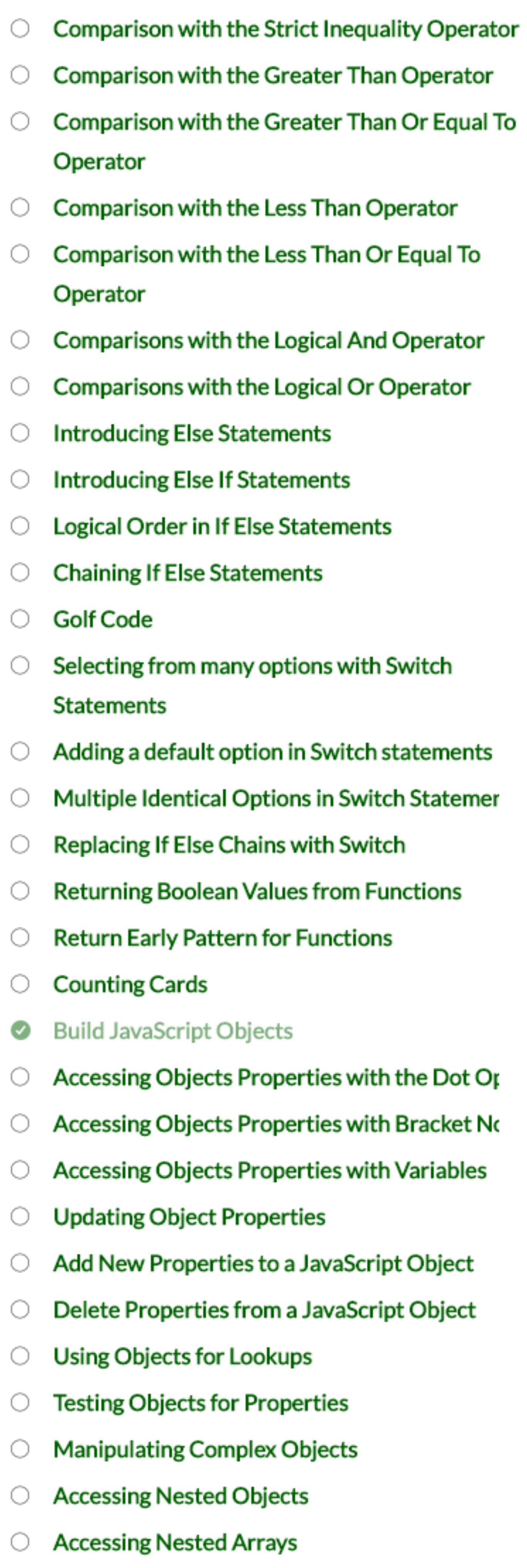


- Comment your JavaScript Code
- Declare JavaScript Variables
- Storing Values with the Assignment Operator
- Initializing Variables with the Assignment Operator
- Understanding Uninitialized Variables
- Understanding Case Sensitivity in Variables
- Add Two Numbers with JavaScript
- Subtract One Number from Another with JavaScript
- Multiply Two Numbers with JavaScript
- Divide One Number by Another with JavaScript
- Increment a Number with JavaScript
- Decrement a Number with JavaScript
- Create Decimal Numbers with JavaScript
- Multiply Two Decimals with JavaScript
- Divide one Decimal by Another with JavaScript
- Finding a Remainder in JavaScript
- Compound Assignment With Augmented Addition
- Compound Assignment With Augmented Subtraction
- Compound Assignment With Augmented Multiplication
- Compound Assignment With Augmented Division
- Convert Celsius to Fahrenheit
- Declare String Variables
- Escaping Literal Quotes in Strings
- Quoting Strings with Single Quotes
- Escape Sequences in Strings
- Concatenating Strings with Plus Operator
- Concatenating Strings with the Plus Equals Operator
- Constructing Strings with Variables
- Appending Variables to Strings

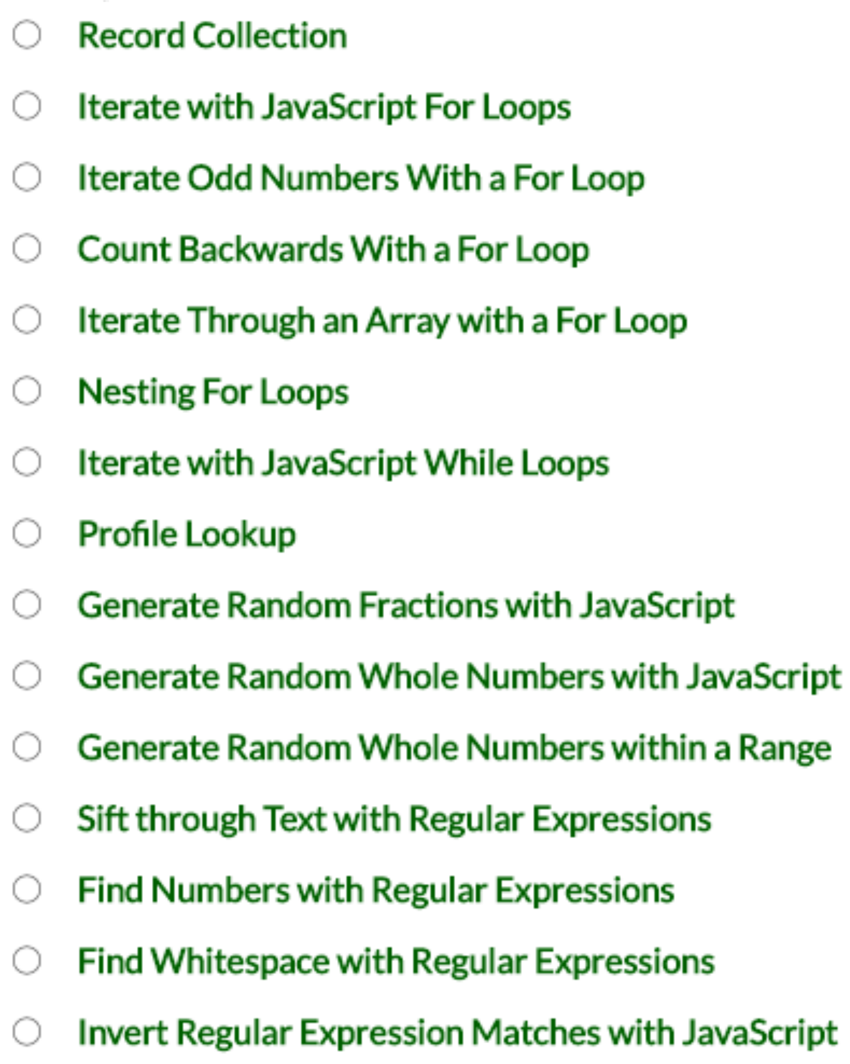


- Use Bracket Notation to Find the First Character in a String
- Understand String Immutability
- Use Bracket Notation to Find the Nth Character in a String
- Use Bracket Notation to Find the Last Character in a String
- Use Bracket Notation to Find the Nth-to-Last Character in a String
- Word Blanks
- Store Multiple Values in one Variable using JavaScript Arrays
- Nest one Array within Another Array
- Access Array Data with Indexes
- Modify Array Data With Indexes
- Access Multi-Dimensional Arrays With Indexes
- Manipulate Arrays With push()
- Manipulate Arrays With pop()
- Manipulate Arrays With shift()
- Manipulate Arrays With unshift()
- Shopping List
- Write Reusable JavaScript with Functions
- Passing Values to Functions with Arguments
- Global Scope and Functions
- Local Scope and Functions
- Global vs. Local Scope in Functions
- Return a Value from a Function with Return
- Assignment with a Returned Value
- Stand in Line
- Understanding Boolean Values
- Use Conditional Logic with If Statements
- Comparison with the Equality Operator
- Comparison with the Strict Equality Operator
- Comparison with the Inequality Operator

Basic Javascript (2)

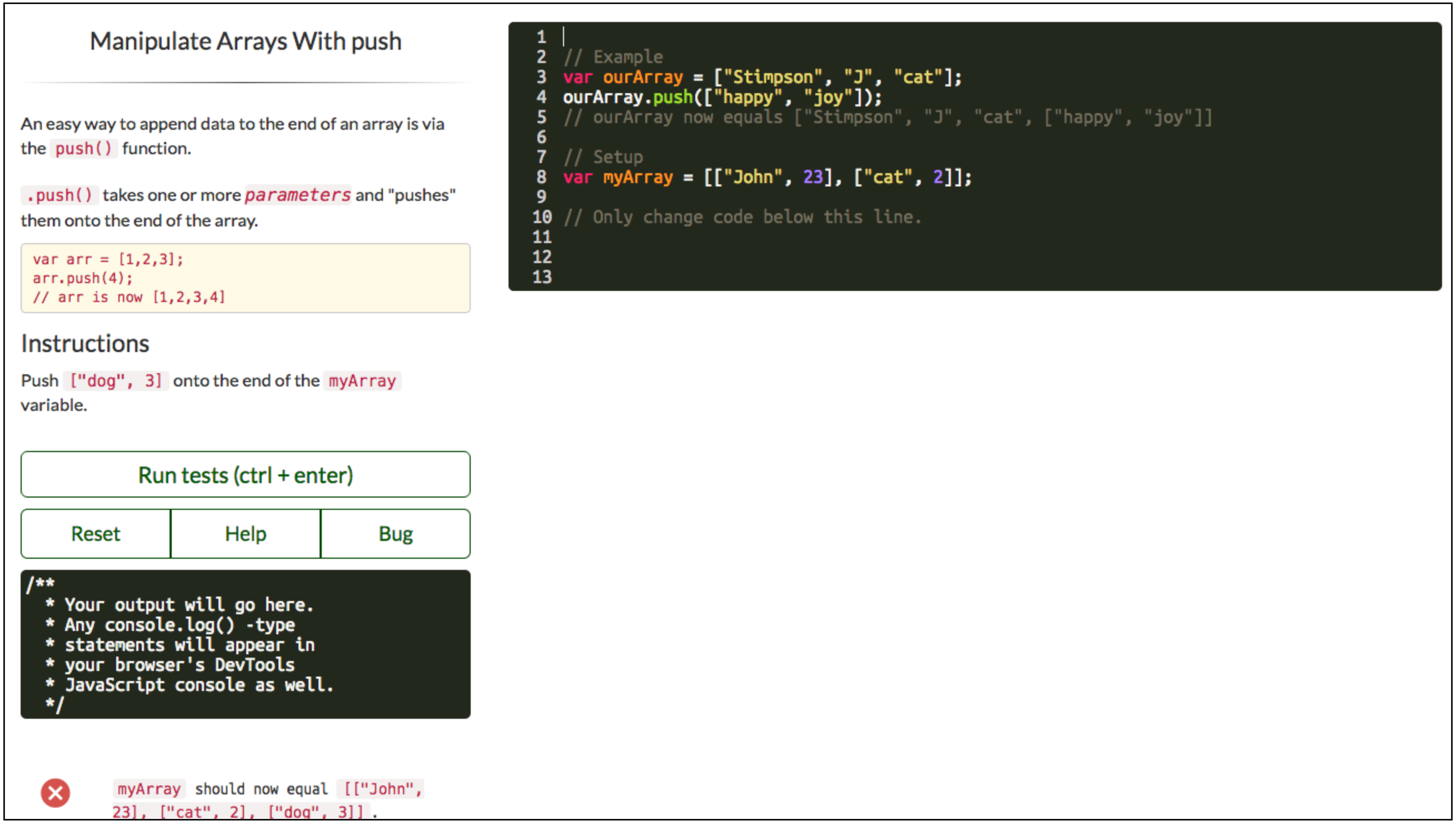


- Comparison with the Strict Inequality Operator
- Comparison with the Greater Than Operator
- Comparison with the Greater Than Or Equal To Operator
- Comparison with the Less Than Operator
- Comparison with the Less Than Or Equal To Operator
- Comparisons with the Logical And Operator
- Comparisons with the Logical Or Operator
- Introducing Else Statements
- Introducing Else If Statements
- Logical Order in If Else Statements
- Chaining If Else Statements
- Golf Code
- Selecting from many options with Switch Statements
- Adding a default option in Switch statements
- Multiple Identical Options in Switch Statemer
- Replacing If Else Chains with Switch
- Returning Boolean Values from Functions
- Return Early Pattern for Functions
- Counting Cards
- ✓ Build JavaScript Objects
- Accessing Objects Properties with the Dot O
- Accessing Objects Properties with Bracket N
- Accessing Objects Properties with Variables
- Updating Object Properties
- Add New Properties to a JavaScript Object
- Delete Properties from a JavaScript Object
- Using Objects for Lookups
- Testing Objects for Properties
- Manipulating Complex Objects
- Accessing Nested Objects
- Accessing Nested Arrays



- Record Collection
- Iterate with JavaScript For Loops
- Iterate Odd Numbers With a For Loop
- Count Backwards With a For Loop
- Iterate Through an Array with a For Loop
- Nesting For Loops
- Iterate with JavaScript While Loops
- Profile Lookup
- Generate Random Fractions with JavaScript
- Generate Random Whole Numbers with JavaScript
- Generate Random Whole Numbers within a Range
- Sift through Text with Regular Expressions
- Find Numbers with Regular Expressions
- Find Whitespace with Regular Expressions
- Invert Regular Expression Matches with JavaScript

Basic Javascript Example



Manipulate Arrays With push

An easy way to append data to the end of an array is via the `push()` function.

`.push()` takes one or more *`parameters`* and "pushes" them onto the end of the array.

```
var arr = [1,2,3];
arr.push(4);
// arr is now [1,2,3,4]
```

Instructions

Push `["dog", 3]` onto the end of the `myArray` variable.

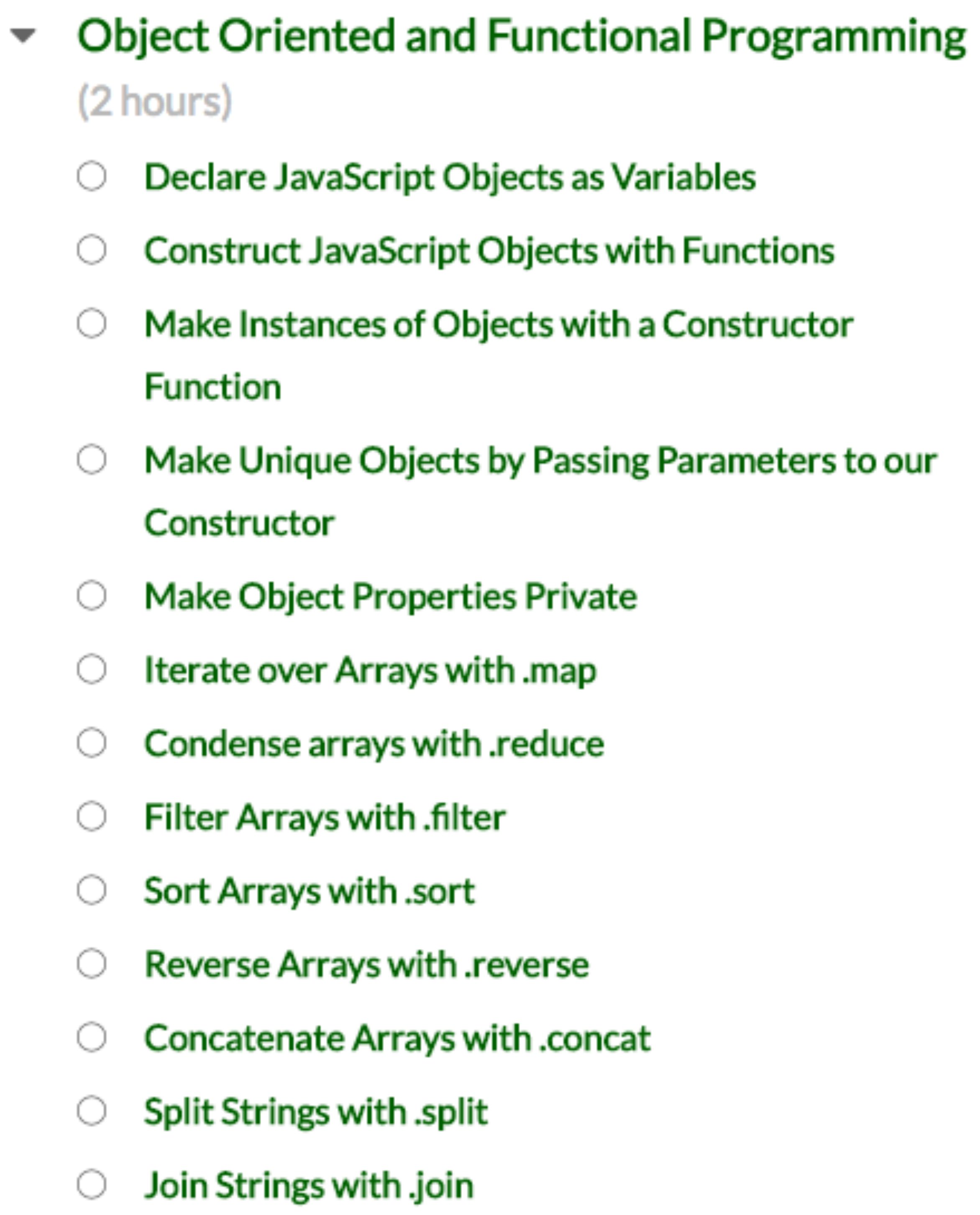
```
/**
 * Your output will go here.
 * Any console.log() -type
 * statements will appear in
 * your browser's DevTools
 * JavaScript console as well.
 */
```

`myArray` should now equal `[["John", 23], ["cat", 2], ["dog", 3]]`.

```

// Example
var ourArray = ["Stimpson", "J", "cat"];
ourArray.push(["happy", "joy"]);
// ourArray now equals ["Stimpson", "J", "cat", ["happy", "joy"]]

// Setup
var myArray = [["John", 23], ["cat", 2]];

// Only change code below this line.



```

Object Oriented & Functional Programming

- 2 hours

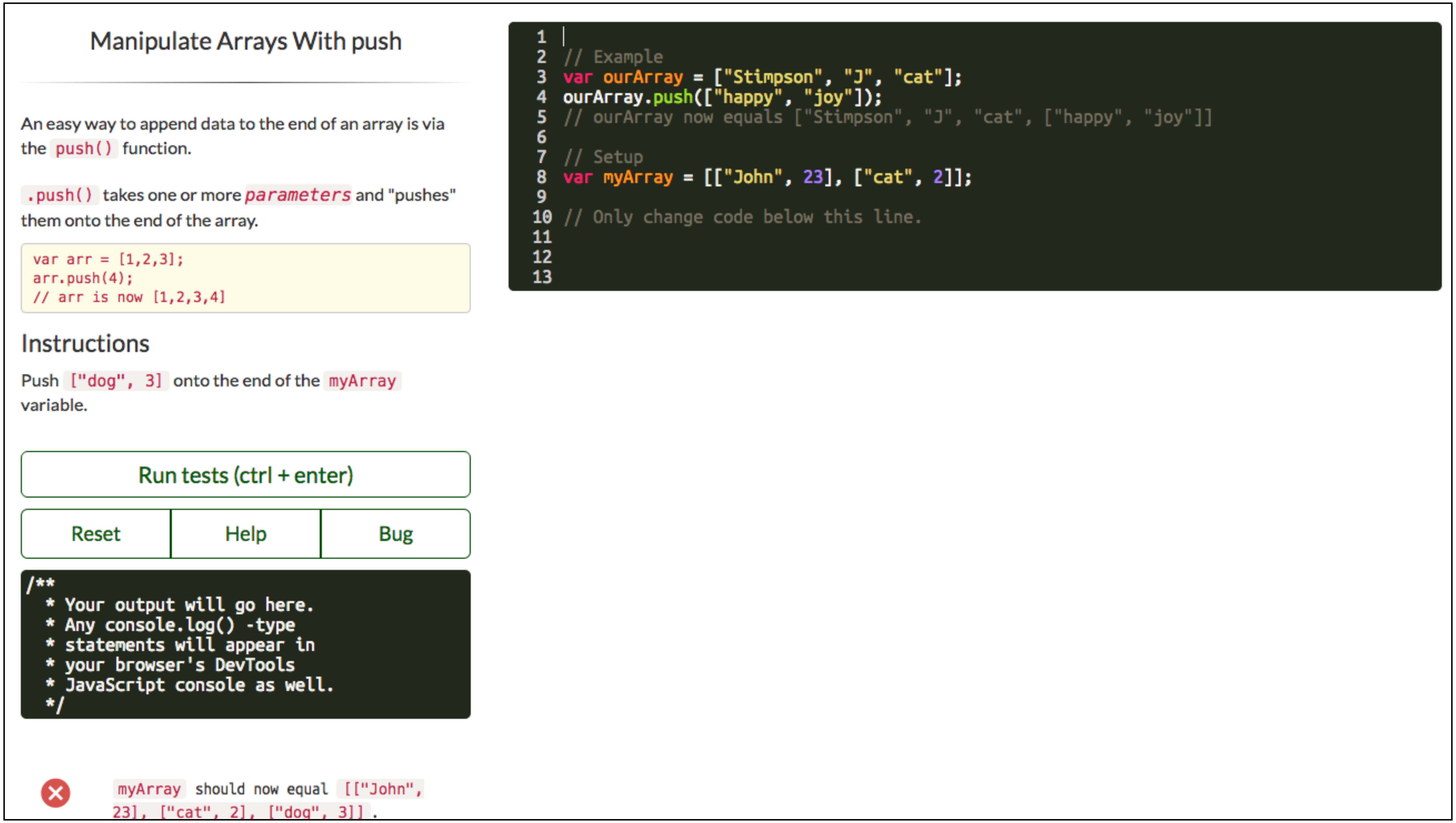


Object Oriented and Functional Programming

(2 hours)

- Declare JavaScript Objects as Variables
- Construct JavaScript Objects with Functions
- Make Instances of Objects with a Constructor Function
- Make Unique Objects by Passing Parameters to our Constructor
- Make Object Properties Private
- Iterate over Arrays with .map
- Condense arrays with .reduce
- Filter Arrays with .filter
- Sort Arrays with .sort
- Reverse Arrays with .reverse
- Concatenate Arrays with .concat
- Split Strings with .split
- Join Strings with .join

Example



Manipulate Arrays With push

An easy way to append data to the end of an array is via the `push()` function.

`.push()` takes one or more *`parameters`* and "pushes" them onto the end of the array.

```
var arr = [1,2,3];
arr.push(4);
// arr is now [1,2,3,4]
```

Instructions

Push `["dog", 3]` onto the end of the `myArray` variable.

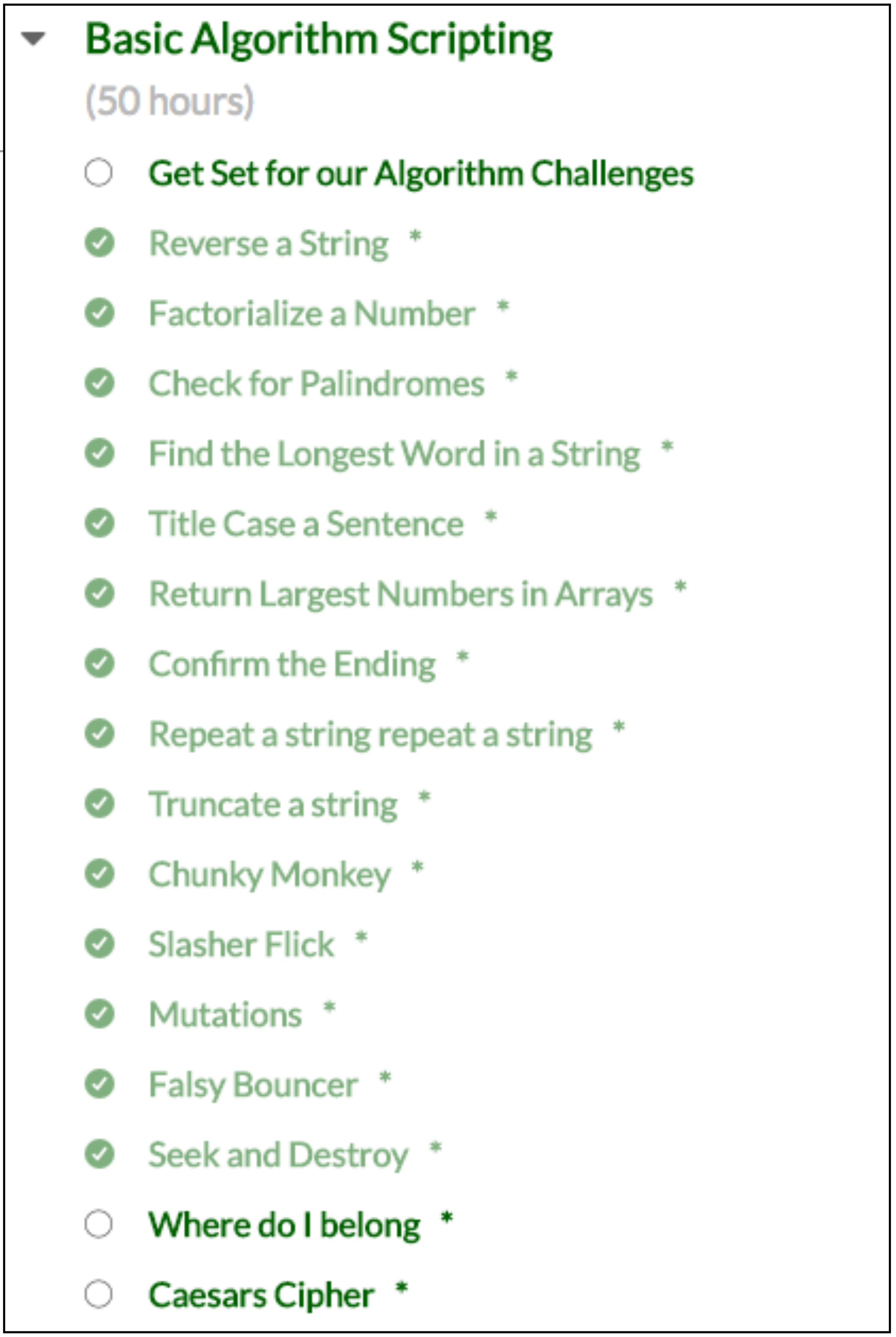
```
/**
 * Your output will go here.
 * Any console.log() -type
 * statements will appear in
 * your browser's DevTools
 * JavaScript console as well.
 */
```

`myArray` should now equal `[["John", 23], ["cat", 2], ["dog", 3]]`.

```

// Example
var ourArray = ["Stimpson", "J", "cat"];
ourArray.push(["happy", "joy"]);
// ourArray now equals ["Stimpson", "J", "cat", ["happy", "joy"]]

// Setup
var myArray = [["John", 23], ["cat", 2]];

// Only change code below this line.



```

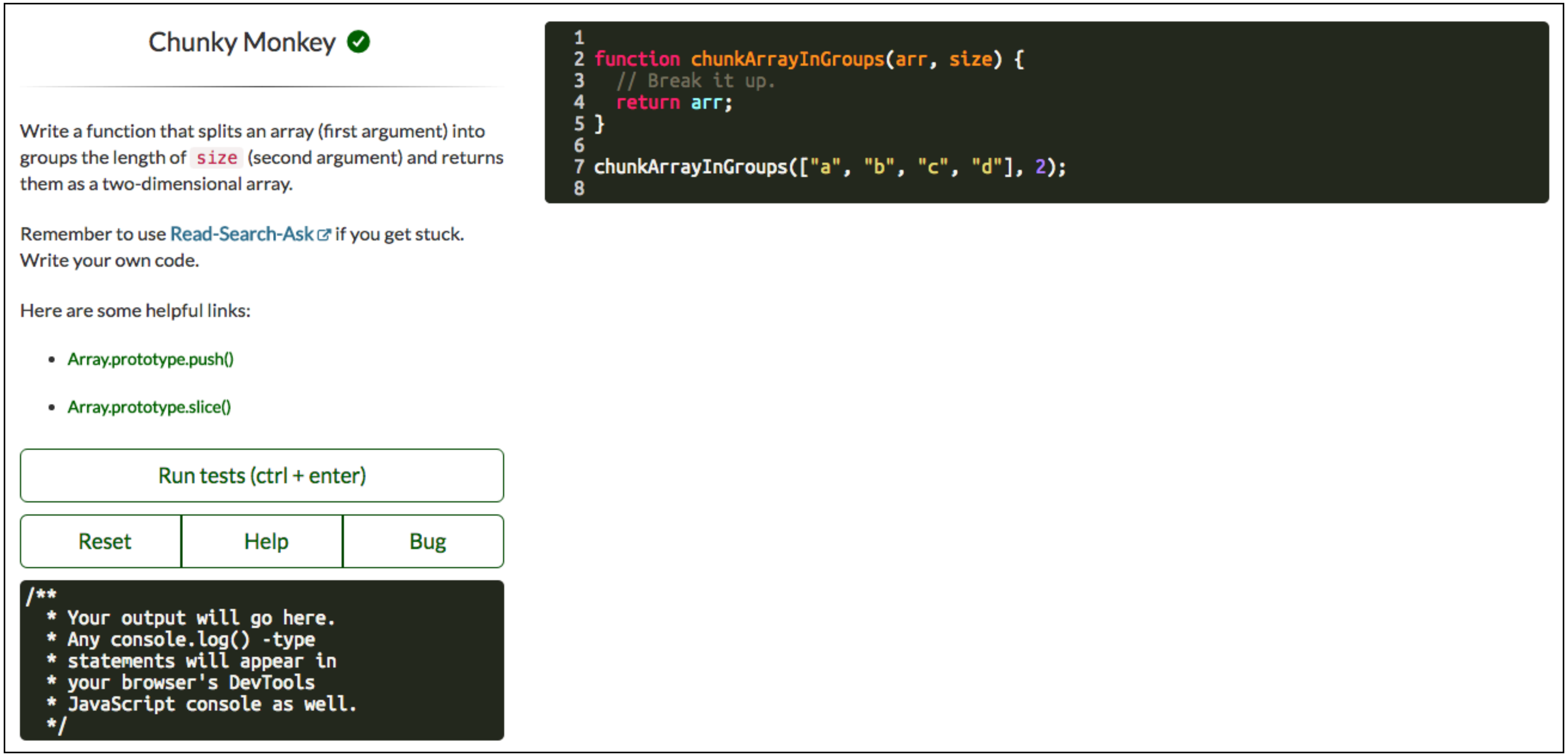
Basic Algorithm Scripting

- 50 Hours


Basic Algorithm Scripting
(50 hours)

- ○ Get Set for our Algorithm Challenges
- ✓ Reverse a String *
- ✓ Factorialize a Number *
- ✓ Check for Palindromes *
- ✓ Find the Longest Word in a String *
- ✓ Title Case a Sentence *
- ✓ Return Largest Numbers in Arrays *
- ✓ Confirm the Ending *
- ✓ Repeat a string repeat a string *
- ✓ Truncate a string *
- ✓ Chunky Monkey *
- ✓ Slasher Flick *
- ✓ Mutations *
- ✓ Falsy Bouncer *
- ✓ Seek and Destroy *
- ○ Where do I belong *
- ○ Caesars Cipher *

Example



Chunky Monkey

Write a function that splits an array (first argument) into groups the length of `size` (second argument) and returns them as a two-dimensional array.

Remember to use Read-Search-Ask if you get stuck. Write your own code.

Here are some helpful links:

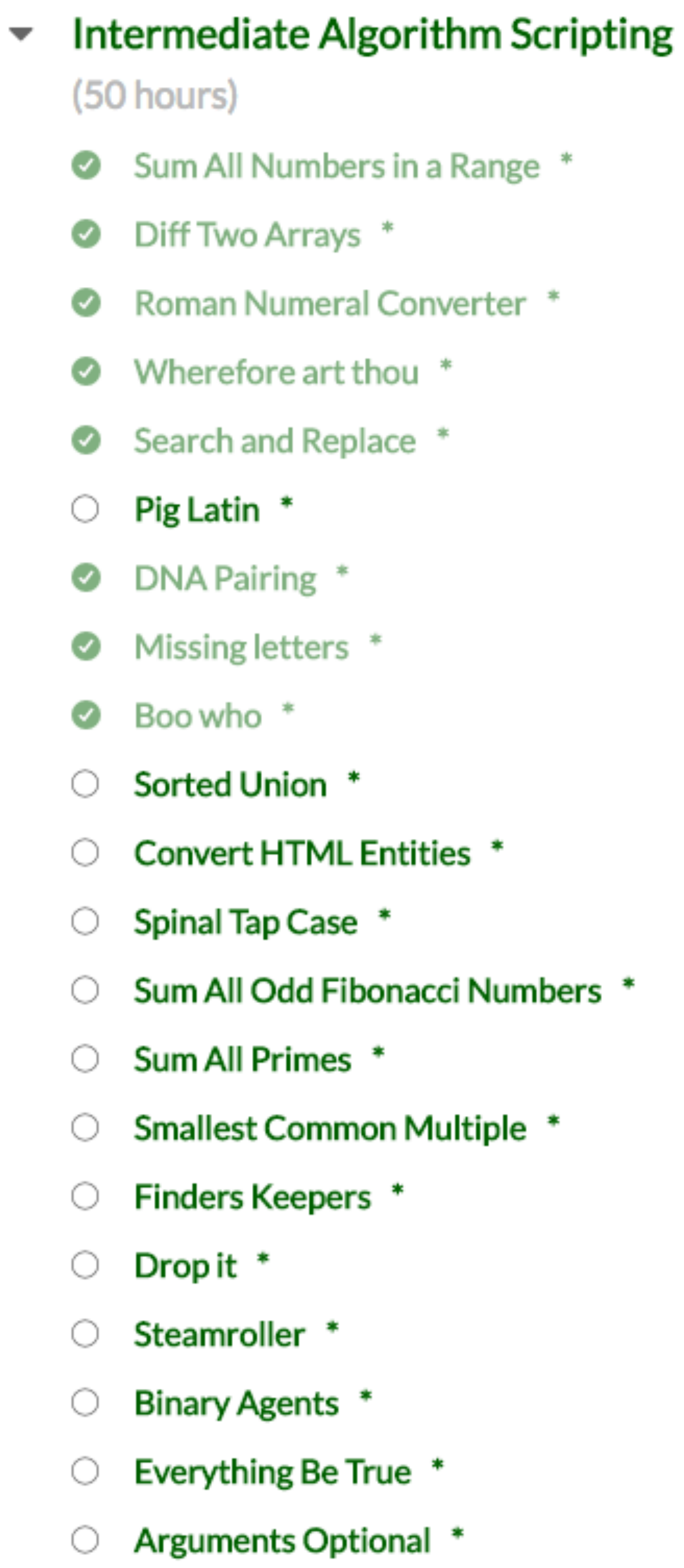
- Array.prototype.push()
- Array.prototype.slice()

```
/**
 * Your output will go here.
 * Any console.log() -type
 * statements will appear in
 * your browser's DevTools
 * JavaScript console as well.
 */
```

```

function chunkArrayInGroups(arr, size) {
  // Break it up.
  return arr;
}

chunkArrayInGroups(["a", "b", "c", "d"], 2);

```

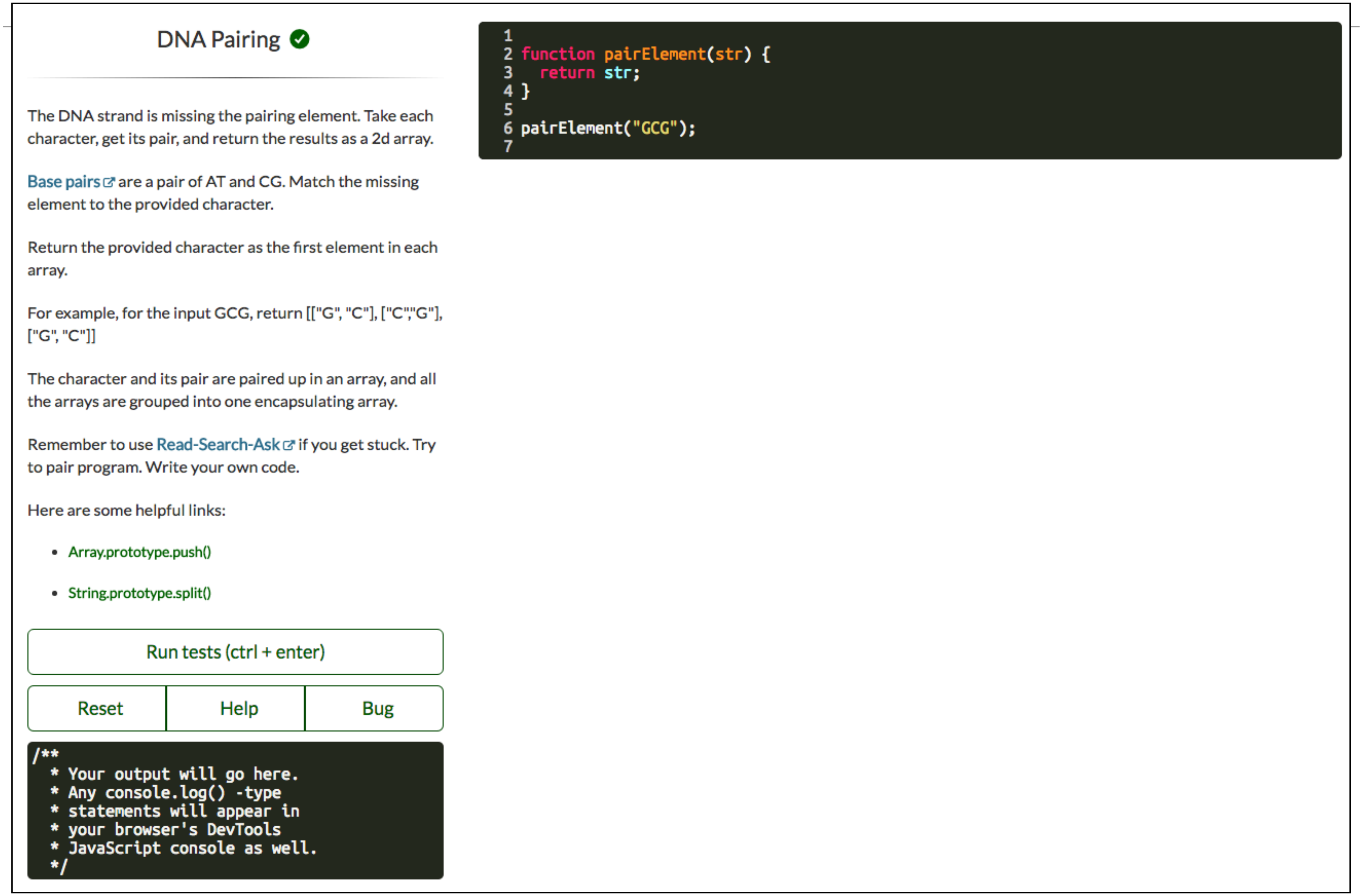
Intermediate Algorithm Scripting

- 50 Hours


Intermediate Algorithm Scripting
(50 hours)

- ✓ Sum All Numbers in a Range *
- ✓ Diff Two Arrays *
- ✓ Roman Numeral Converter *
- ✓ Wherefore art thou *
- ✓ Search and Replace *
- ○ Pig Latin *
- ✓ DNA Pairing *
- ✓ Missing letters *
- ✓ Boo who *
- ○ Sorted Union *
- ○ Convert HTML Entities *
- ○ Spinal Tap Case *
- ○ Sum All Odd Fibonacci Numbers *
- ○ Sum All Primes *
- ○ Smallest Common Multiple *
- ○ Finders Keepers *
- ○ Drop it *
- ○ Steamroller *
- ○ Binary Agents *
- ○ Everything Be True *
- ○ Arguments Optional *

Example



DNA Pairing

The DNA strand is missing the pairing element. Take each character, get its pair, and return the results as a 2d array.

Base pairs are a pair of AT and CG. Match the missing element to the provided character.

Return the provided character as the first element in each array.

For example, for the input GCG, return [["G", "C"], ["C","G"], ["G", "C"]]

The character and its pair are paired up in an array, and all the arrays are grouped into one encapsulating array.

Remember to use Read-Search-Ask if you get stuck. Try to pair program. Write your own code.

Here are some helpful links:

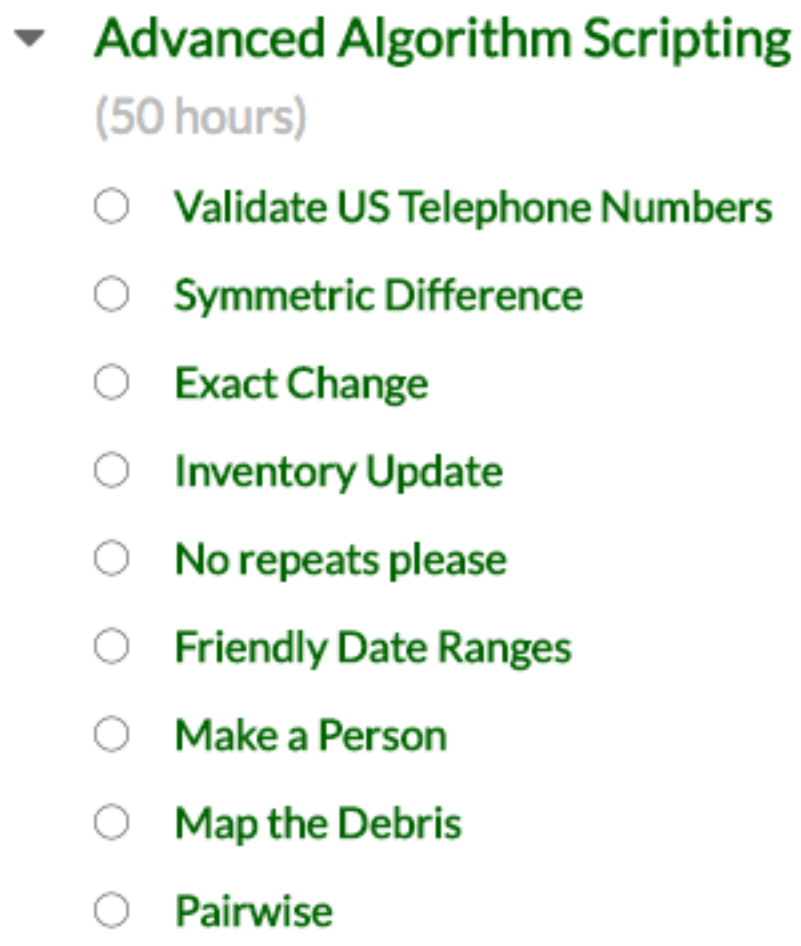
- Array.prototype.push()
- String.prototype.split()

```
/**
 * Your output will go here.
 * Any console.log() -type
 * statements will appear in
 * your browser's DevTools
 * JavaScript console as well.
 */
```

```javascript

function pairElement(str) {
  return str;
}

pairElement("GCG");

```

Advanced Algorithm Scripting

- 50 Hours

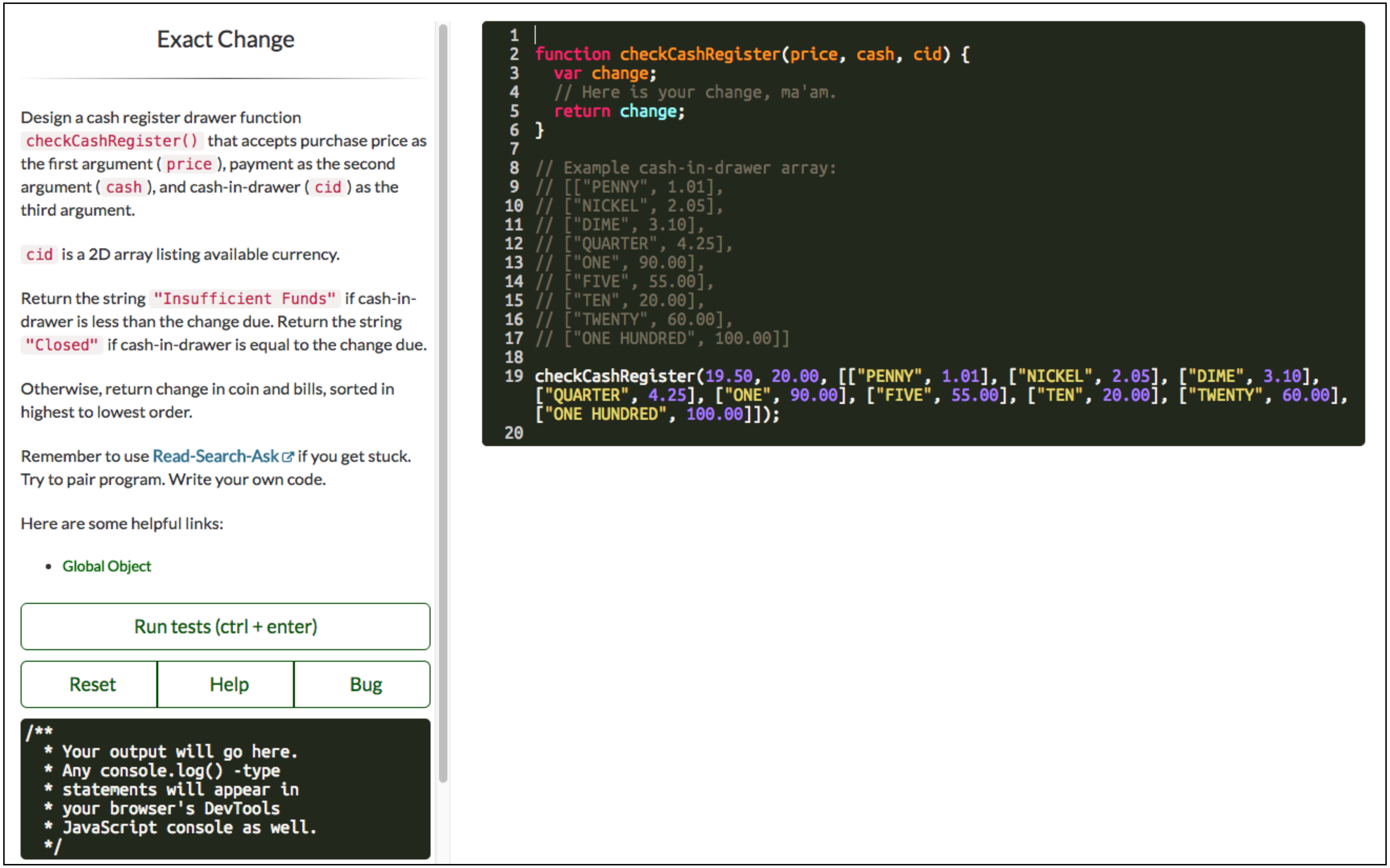


Advanced Algorithm Scripting

(50 hours)

- Validate US Telephone Numbers
- Symmetric Difference
- Exact Change
- Inventory Update
- No repeats please
- Friendly Date Ranges
- Make a Person
- Map the Debris
- Pairwise

Example



Exact Change

Design a cash register drawer function `checkCashRegister()` that accepts purchase price as the first argument (`price`), payment as the second argument (`cash`), and cash-in-drawer (`cid`) as the third argument.

`cid` is a 2D array listing available currency.

Return the string `"Insufficient Funds"` if cash-in-drawer is less than the change due. Return the string `"Closed"` if cash-in-drawer is equal to the change due.

Otherwise, return change in coin and bills, sorted in highest to lowest order.

Remember to use Read-Search-Ask if you get stuck. Try to pair program. Write your own code.

Here are some helpful links:

- Global Object

```
/**
 * Your output will go here.
 * Any console.log() -type
 * statements will appear in
 * your browser's DevTools
 * JavaScript console as well.
 */
```

```javascript

function checkCashRegister(price, cash, cid) {
  var change;
  // Here is your change, ma'am.
  return change;
}

// Example cash-in-drawer array:
// [["PENNY", 1.01],
// ["NICKEL", 2.05],
// ["DIME", 3.10],
// ["QUARTER", 4.25],
// ["ONE", 90.00],
// ["FIVE", 55.00],
// ["TEN", 20.00],
// ["TWENTY", 60.00],
// ["ONE HUNDRED", 100.00]]

checkCashRegister(19.50, 20.00, [["PENNY", 1.01], ["NICKEL", 2.05], ["DIME", 3.10],
["QUARTER", 4.25], ["ONE", 90.00], ["FIVE", 55.00], ["TEN", 20.00], ["TWENTY", 60.00],
["ONE HUNDRED", 100.00]]);

```