Algorithms in Javascript

JavaScript Skills - FreeCodeCamp

Learn to code for free.

Join a supportive community of millions of coders.

Build projects and earn free certifications.

Get experience by coding for nonprofits.

HTML5

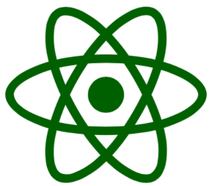
CSS3

JavaScript

Databases

Git & GitHub

Node.js

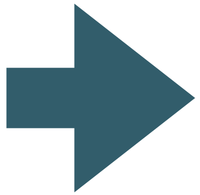


React.js

D3.js

Curriculum

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



- Responsive Web Design Certification (300 hours)
- Javascript Algorithms And Data Structures Certification (300 hours)
- Front End Libraries Certification (300 hours)
- Data Visualization Certification (300 hours)
- Apis And Microservices Certification (300 hours)
- Information Security And Quality Assurance Certification (300 hours)
- Coding Interview Prep (Thousands of hours of challenges)

Javascript Algorithms & Data Structures

- Javascript Algorithms And Data Structures Certification (300 hours)
  - Basic JavaScript
  - ES6
  - Regular Expressions
  - Debugging
  - Basic Data Structures
  - Basic Algorithm Scripting
  - Object Oriented Programming
  - Functional Programming
  - Intermediate Algorithm Scripting
  - JavaScript Algorithms and Data Structures Projects

Basic Javascript

- [ ] 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
- [ ] 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
- [ ] Find the Length of a String
- [ ] 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
- [ ] Understanding Undefined Value returned from a Function
- [ ] 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
- [ ] Practice comparing different values
- [ ] Comparison with the Inequality Operator
- [ ] 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
- [ ] Adding a Default Option in Switch Statements
- [ ] Multiple Identical Options in Switch Statements
- [ ] Replacing If Else Chains with Switch
- [ ] Returning Boolean Values from Functions
- [ ] Return Early Pattern for Functions
- [ ] Counting Cards
- [x] Build JavaScript Objects
- [ ] Accessing Object Properties with Dot Notation
- [ ] Accessing Object Properties with Bracket Notation
- [ ] Accessing Object 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 While Loops
- [ ] 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 Do...While Loops
- [ ] Profile Lookup
- [ ] Generate Random Fractions with JavaScript
- [ ] Generate Random Whole Numbers with JavaScript
- [ ] Generate Random Whole Numbers within a Range
- [ ] Use the parseInt Function
- [ ] Use the parseInt Function with a Radix
- [ ] Use the Conditional (Ternary) Operator
- [ ] Use Multiple Conditional (Ternary) Operators

Basic Javascript Example

Basic JavaScript: 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]
```

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

Run the Tests

Reset All Code

Get a hint

Ask for help

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

```
1  // Example
2  var ourArray = ["Stimpson", "J", "cat"];
3  ourArray.push(["happy", "joy"]);
4  // ourArray now equals ["Stimpson", "J", "cat", ["happy", "joy"]]
5
6  // Setup
7  var myArray = [["John", 23], ["cat", 2]];
8
9  // Only change code below this line.
10
11
```

```
/**
* Your test output will go here.
*/
```

ES6

- Explore Differences Between the var and let Keywords
- Compare Scopes of the var and let Keywords
- Declare a Read-Only Variable with the const Keyword
- Mutate an Array Declared with const
- Prevent Object Mutation
- Use Arrow Functions to Write Concise Anonymous Functions
- Write Arrow Functions with Parameters
- Write Higher Order Arrow Functions
- Set Default Parameters for Your Functions
- Use the Rest Operator with Function Parameters
- Use the Spread Operator to Evaluate Arrays In-Place
- Use Destructuring Assignment to Assign Variables from Objects
- Use Destructuring Assignment to Assign Variables from Nested Objects
- Use Destructuring Assignment to Assign Variables from Arrays
- Use Destructuring Assignment with the Rest Operator to Reassign Array Elements
- Use Destructuring Assignment to Pass an Object as a Function's Parameters
- Create Strings using Template Literals
- Write Concise Object Literal Declarations Using Simple Fields
- Write Concise Declarative Functions with ES6
- Use class Syntax to Define a Constructor Function
- Use getters and setters to Control Access to an Object
- Understand the Differences Between import and require
- Use export to Reuse a Code Block
- Use * to Import Everything from a File
- Create an Export Fallback with export default
- Import a Default Export

ES6: Use Arrow Functions to Write Concise Anonymous Functions

In JavaScript, we often don't need to name our functions, especially when passing a function as an argument to another function. Instead, we create inline functions. We don't need to name these functions because we do not reuse them anywhere else.

To achieve this, we often use the following syntax:

```
const myFunc = function() {
  const myVar = "value";
  return myVar;
}
```

ES6 provides us with the syntactic sugar to not have to write anonymous functions this way. Instead, you can use **arrow function syntax**:

```
const myFunc = () => {
  const myVar = "value";
  return myVar;
}
```

When there is no function body, and only a return value, arrow function syntax allows you to omit the keyword `return` as well as the brackets surrounding the code. This helps simplify smaller functions into one-line statements:

```
const myFunc = () => "value"
```

This code will still return `value` by default.

Rewrite the function assigned to the variable `magic` which returns a new `Date()` to use arrow function syntax. Also make sure nothing is defined using the keyword `var`.

Run the Tests

```
1 var magic = function() {
2   "use strict";
3   return new Date();
4 };
```

```
/**
* Your test output will go here.
*/
```

Basic Data Structures

- Use an Array to Store a Collection of Data
- Access an Array's Contents Using Bracket Notation
- Add Items to an Array with push() and unshift()
- Remove Items from an Array with pop() and shift()
- Remove Items Using splice()
- Add Items Using splice()
- Copy Array Items Using slice()
- Copy an Array with the Spread Operator
- Combine Arrays with the Spread Operator
- Check For The Presence of an Element With indexOf()
- Iterate Through All an Array's Items Using For Loops
- Create complex multi-dimensional arrays
- Add Key-Value Pairs to JavaScript Objects
- Modify an Object Nested Within an Object
- Access Property Names with Bracket Notation
- Use the delete Keyword to Remove Object Properties
- Check if an Object has a Property
- Iterate Through the Keys of an Object with a for...in Statement
- Generate an Array of All Object Keys with Object.keys()
- Modify an Array Stored in an Object

Basic Data Structures: Copy Array Items Using slice()

The next method we will cover is `slice()`. `slice()`, rather than modifying an array, copies, or *extracts*, a given number of elements to a new array, leaving the array it is called upon untouched. `slice()` takes only 2 parameters — the first is the index at which to begin extraction, and the second is the index at which to stop extraction (extraction will occur up to, but not including the element at this index). Consider this:

```
let weatherConditions = ['rain', 'snow', 'sleet', 'hail', 'clear'];

let todaysWeather = weatherConditions.slice(1, 3);
// todaysWeather equals ['snow', 'sleet'];
// weatherConditions still equals ['rain', 'snow', 'sleet', 'hail',
'clear']
```

In effect, we have created a new array by extracting elements from an existing array.

We have defined a function, `forecast`, that takes an array as an argument. Modify the function using `slice()` to extract information from the argument array and return a new array that contains the elements `'warm'` and `'sunny'`.

Run the Tests

Reset All Code

Get a hint

Ask for help

`forecast` should return `["warm", "sunny"]`

The `forecast` function should utilize the `slice()` method

```
function forecast(arr) {
  // change code below this line

  return arr;
}

// do not change code below this line
console.log(forecast(['cold', 'rainy', 'warm', 'sunny', 'cool', 'thunderstorms']));
```

```
cold,rainy,warm,sunny,cool,thunderstorms
```

Basic Algorithm Scripting

- [ ] Convert Celsius to Fahrenheit
- [x] Reverse a String
- [x] Factorialize a Number
- [x] Find the Longest Word in a String
- [x] Return Largest Numbers in Arrays
- [x] Confirm the Ending
- [x] Repeat a String Repeat a String
- [x] Truncate a String
- [ ] Finders Keepers
- [x] Boo who
- [x] Title Case a Sentence
- [ ] Slice and Splice
- [x] Falsy Bouncer
- [ ] Where do I Belong
- [x] Mutations
- [x] Chunky Monkey

Object Oriented Programming

- [ ] Create a Basic JavaScript Object
- [ ] Use Dot Notation to Access the Properties of an Object
- [ ] Create a Method on an Object
- [ ] Make Code More Reusable with the this Keyword
- [ ] Define a Constructor Function
- [ ] Use a Constructor to Create Objects
- [ ] Extend Constructors to Receive Arguments
- [ ] Verify an Object's Constructor with instanceof
- [ ] Understand Own Properties
- [ ] Use Prototype Properties to Reduce Duplicate Code
- [ ] Iterate Over All Properties
- [ ] Understand the Constructor Property
- [ ] Change the Prototype to a New Object
- [ ] Remember to Set the Constructor Property when Changing the Prototype
- [ ] Understand Where an Object’s Prototype Comes From
- [ ] Understand the Prototype Chain
- [ ] Use Inheritance So You Don't Repeat Yourself
- [ ] Inherit Behaviors from a Supertype
- [ ] Set the Child's Prototype to an Instance of the Parent
- [ ] Reset an Inherited Constructor Property
- [ ] Add Methods After Inheritance
- [ ] Override Inherited Methods
- [ ] Use a Mixin to Add Common Behavior Between Unrelated Objects
- [ ] Use Closure to Protect Properties Within an Object from Being Modified Externally
- [ ] Understand the Immediately Invoked Function Expression (IIFE)
- [ ] Use an IIFE to Create a Module

Intermediate Algorithm Scripting

- [x] Sum All Numbers in a Range
- [x] Diff Two Arrays
- [x] Seek and Destroy
- [x] Wherefore art thou
- [ ] Spinal Tap Case
- [ ] Pig Latin
- [x] Search and Replace
- [x] DNA Pairing
- [x] Missing letters
- [ ] Sorted Union
- [ ] Convert HTML Entities
- [ ] Sum All Odd Fibonacci Numbers
- [ ] Sum All Primes
- [ ] Smallest Common Multiple
- [ ] Drop it
- [ ] Steamroller
- [ ] Binary Agents
- [ ] Everything Be True
- [ ] Arguments Optional
- [ ] Make a Person
- [ ] Map the Debris

Intermediate Algorithm Scripting: 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.

Run the Tests

Reset All Code

Get a hint

Ask for help

- `pairElement("ATCGA")` should return `[["A","T"],["T","A"],["C","G"],["G","C"],["A","T"]]`.
- `pairElement("TTGAG")` should return `[["T","A"],["T","A"],["G","C"],["A","T"],["G","C"]]`.
- `pairElement("CTCTA")` should return `[["C","G"],["T","A"],["C","G"],["T","A"],["A","T"]]`.

```
1 function pairElement(str) {
2   return str;
3 }
4
5 pairElement("GCG");
```

```
/**
* Your test output will go here.
*/
```

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. We already. have the route in place:

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

and a matching handler - which you will have to enhance.

```
  signup: {
    handler: function(request, h) {
      return h.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 1

- 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.

index.js

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

app/controllers/accounts.js

```
  signup: {
    handler: function(request, h) {
      const user = request.payload;
      this.users[user.email] = user;
      return h.redirect('/home');
    }
  },
```

Look up user in login

```
login: {
  handler: function(request, h) {
    const user = request.payload;
    if ((user.email in this.users) && (user.password === this.users[user.email].password)) {
      return h.redirect('/home');
    }
    return h.redirect('/');
  }
},
```

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

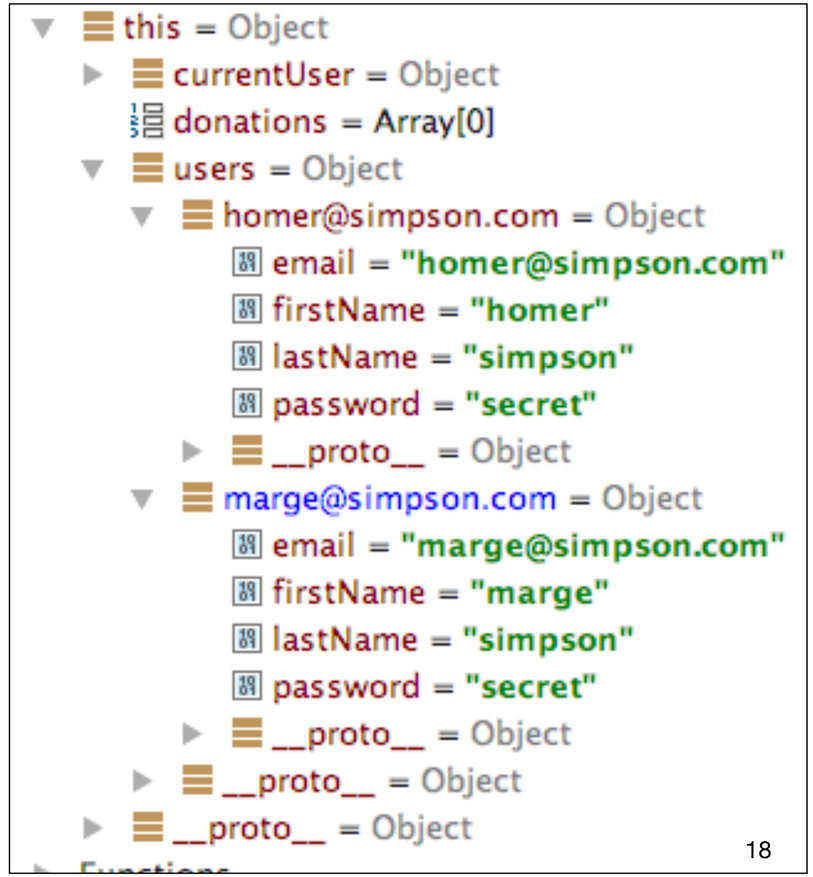
Preload users

- initUsers an object literal
- It contains 2 name/value pairs
  - Name is an email of a user
  - Value is an user 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: [],
});
```

```
signup: {
  handler: function(request, h) {
    const user = request.payload;
    this.users[user.email] = user;
    return h.redirect('/home');
  }
},
```



Current User Tracking

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

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

Donor

```
exports.donate = {

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

};
```

```
...
          <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>
...
```