

Data Types and How to Master Them

Arrays, Objects, and Loops - Oh my!

Why are we here today?

Who is Thinkful



Update with thoughtful verbage & styling

Why Learn This?

- Seldom is our data exactly how we want it
- Learning the more powerful features of JS can make us more efficient
- A critical skill for any developer

How this works

bit.ly/tf-array-start-made-up-link

What is a variable in Javascript

```
var firstVariable = 20
```

Array Basics

- Arrays are lists
- They can contain all sorts of data - variables, strings, objects, even functions
- They are zero-indexed (start at 0)

```
let arr = [1, 2, 3]
```

```
arr[0] // 1
```

```
arr[2] // 3
```

Array Drills

```
JS
1 // Array basics
2 let groceryList = ['apples', 'bananas', 'coffee']
3
4 // Access values from our array
5 //   values start at 0
6 console.log(groceryList[0])
7
8 let myName = 'Chris'
9 let randomList = [myName, { x: 1, y: 2 }, 'a string',
10   are neat! }]
11 console.log(randomList[3]())
12
13 // Array.length returns length of array
14 //   3 items in the array, returns 3
15 console.log(groceryList.length)
```


For loops

- The most common and powerful type of loop
- Each for loop has 3 essential elements
 - Starting value
 - Ending condition
 - Incrementer (or decrementer)

```
let arr = [1, 2, 3]

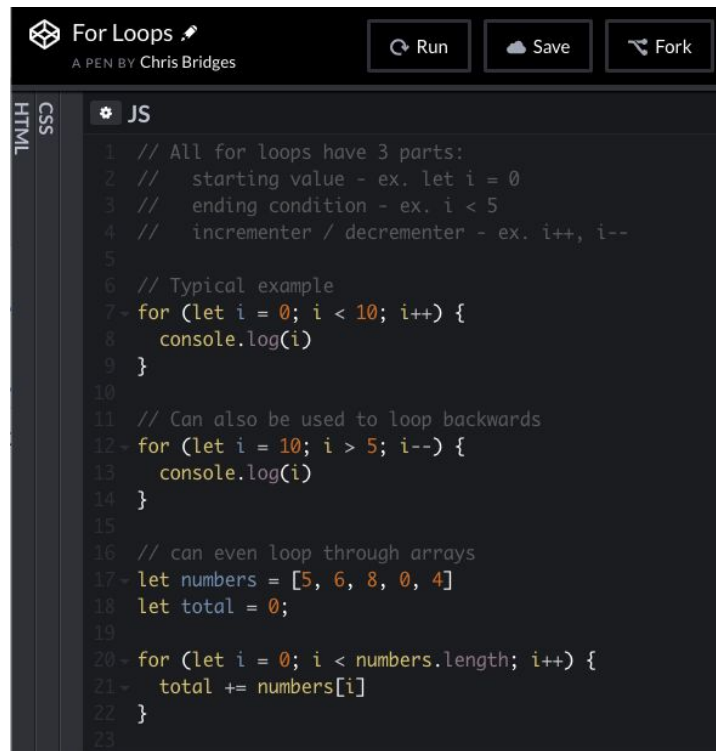
for (let i = 0; i < arr.length; i++) {

    console.log(i)

}

// 1, 2, 3
```

For Loop Drills



The screenshot shows a code editor interface for a project titled "For Loops" by Chris Bridges. The editor has tabs for HTML, CSS, and JS, with the JS tab selected. The code contains several comments and examples of for loops, including a typical forward loop, a backward loop, and a loop through an array.

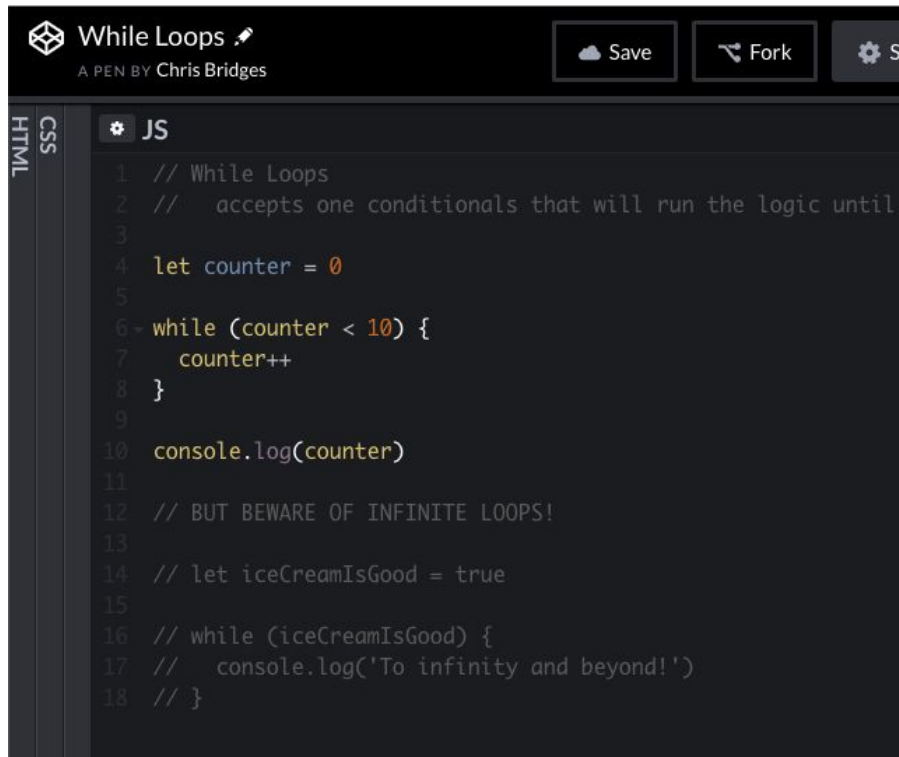
```
1 // All for loops have 3 parts:
2 //   starting value - ex. let i = 0
3 //   ending condition - ex. i < 5
4 //   incremter / decremter - ex. i++, i--
5
6 // Typical example
7 - for (let i = 0; i < 10; i++) {
8   console.log(i)
9 }
10
11 // Can also be used to loop backwards
12 - for (let i = 10; i > 5; i--) {
13   console.log(i)
14 }
15
16 // can even loop through arrays
17 - let numbers = [5, 6, 8, 0, 4]
18   let total = 0;
19
20 - for (let i = 0; i < numbers.length; i++) {
21   total += numbers[i]
22 }
23
```

While loops

- Will continue looping until a certain condition is met
- Used when we do not know exactly how many times we would like our loop to run
- Beware of infinite loops!

```
let str = ''  
  
while (str.length < 5) {  
  
    str += 'a'  
  
}  
  
// str === 'aaaaa'
```

While Loop Drills



The screenshot shows a code editor interface with a dark theme. At the top, the title bar reads "While Loops" with a small icon and "A PEN BY Chris Bridges". To the right of the title bar are three buttons: "Save" (with a cloud icon), "Fork" (with a fork icon), and a settings gear icon. On the left side, there is a vertical sidebar with three tabs: "HTML", "CSS", and "JS". The "JS" tab is selected and highlighted. The main code area displays the following JavaScript code:

```
1 // While Loops
2 //   accepts one conditionals that will run the logic until
3
4 let counter = 0
5
6 while (counter < 10) {
7   counter++
8 }
9
10 console.log(counter)
11
12 // BUT BEWARE OF INFINITE LOOPS!
13
14 // let iceCreamIsGood = true
15
16 // while (iceCreamIsGood) {
17 //   console.log('To infinity and beyond!')
18 // }
```

Array.forEach() && Array.map()

- `forEach` directly manipulates the array it is called upon
- `map` creates a copy of the array, then manipulates

```
let myArr = ['a', 'b', 'c']
```

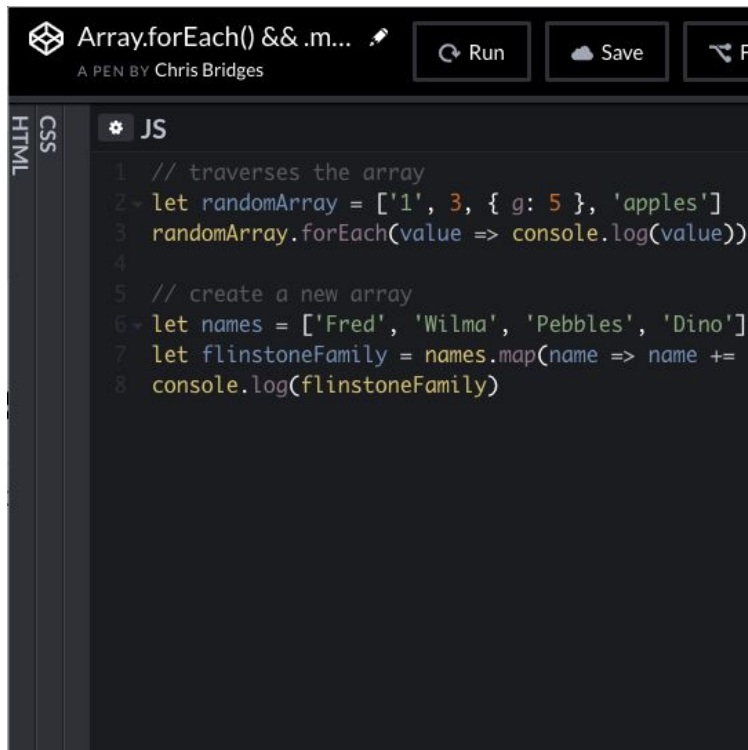
```
myArr.forEach((value, index) => {
```

```
    myArr[index] = value + ' new value'
```

```
})
```

```
// myArr -> ['a new value', 'b new value', 'c  
new value']
```

Array.forEach() & Array.map() Drills



The screenshot shows a code editor interface with a dark theme. At the top, there's a title bar with a logo, the text "Array.forEach() & .m...", and a "Run" button. Below the title bar, there's a sidebar with "HTML", "CSS", and "JS" tabs, with "JS" being the active tab. The main editor area contains the following JavaScript code:

```
1 // traverses the array
2 let randomArray = ['1', 3, { g: 5 }, 'apples']
3 randomArray.forEach(value => console.log(value))
4
5 // create a new array
6 let names = ['Fred', 'Wilma', 'Pebbles', 'Dino']
7 let flinstoneFamily = names.map(name => name +=
8 console.log(flinstoneFamily))
```

Array.reduce()

- Used to condense an entire array down to one single value
- Can be useful for summing an array of numbers or finding the average value of that array
- Each reduce operation has an accumulator and current value argument

```
let numbers = [1, 2, 3]
```

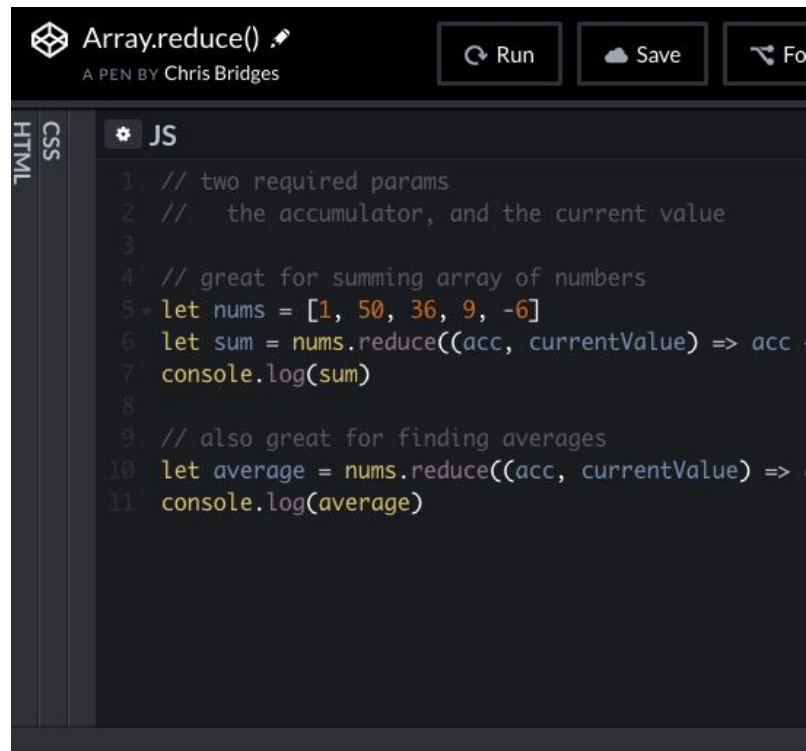
```
let sumOfNumbers = numbers.reduce((acc,  
value) => {
```

```
    return acc + value
```

```
})
```

```
// 6
```

Array.reduce() Drills



The screenshot shows a code editor interface for a Pen titled "Array.reduce()" by Chris Bridges. The editor has tabs for HTML, CSS, and JS. The JS tab is active, displaying a JavaScript snippet that demonstrates the use of the `Array.reduce()` method. The code includes comments explaining the required parameters and provides two examples: summing an array of numbers and finding the average of an array.

```
1 // two required params
2 //   the accumulator, and the current value
3
4 // great for summing array of numbers
5 let nums = [1, 50, 36, 9, -6]
6 let sum = nums.reduce((acc, currentValue) => acc + currentValue, 0)
7 console.log(sum)
8
9 // also great for finding averages
10 let average = nums.reduce((acc, currentValue) => acc + currentValue, 0) / nums.length
11 console.log(average)
```


Array.filter()

- Loops through array and filters out values that do not meet criteria

```
let myArr = ['a', 'b', 'c']
```

```
let newArr = myArr.filter(value => value ===  
  'b')
```

```
// newArr -> ['b']
```

Array.filter() Drills



The screenshot shows a code editor interface for a Pen titled "Array.filter()" by Chris Bridges. The editor has tabs for HTML, CSS, and JS. The JS tab is active, displaying the following code:

```
1 // can be used to filter out words of a certain length
2 let words = ['programming', 'JavaScript', 'binary', 'termina
3 let longerWords = words.filter(word => word.length > 5)
4 console.log(longerWords)
5
6 // can be used to filter out numbers
7 let numbers = [6, 4, 100, -8, 72, 87, -1, 0, 21]
8 let smallNumbers = numbers.filter(number => number < 10)
9 console.log(smallNumbers)
10
11 // or return only even numbers
12 let evenNumbers = numbers.filter(number => number % 2 === 0)
13 console.log(evenNumbers)]
```

Push / Pop / Shift / Unshift

- Useful shorthands for a variety of common array manipulations
 - Push - adds value to end of array
 - Pop - removes value from end of array
 - Shift - removes first element from array
 - Unshift - inserts element to beginning of array

Push / Pop / Shift / Unshift



The screenshot shows a code editor window titled "Array Push / Pop / Shift / Unshift" by Chris Bridges. The editor contains JavaScript code demonstrating four array methods: push, pop, shift, and unshift. The code is as follows:

```
1 let myArray = ['First Value', 'Second Value', -10]
2
3 // Push - adds value to end of array
4 myArray.push('Added to the end')
5 console.log(myArray)
6
7 // Pop - removes last value from array
8 myArray.pop()
9 console.log(myArray)
10
11 // Shift - removes first element from an array
12 myArray.shift()
13 console.log(myArray)
14
15 // Unshift - adds element to beginning of array
16 myArray.unshift('Added using unshift')
17 console.log(myArray)
```

Array.slice()

- Creates a copy of array with values from indexes you specify
 - First argument is inclusive (will be included in your new array)
 - Second argument is exclusive (will be excluded from your new array)

```
let myArr = [1, 2, 3, 4, 5]
```

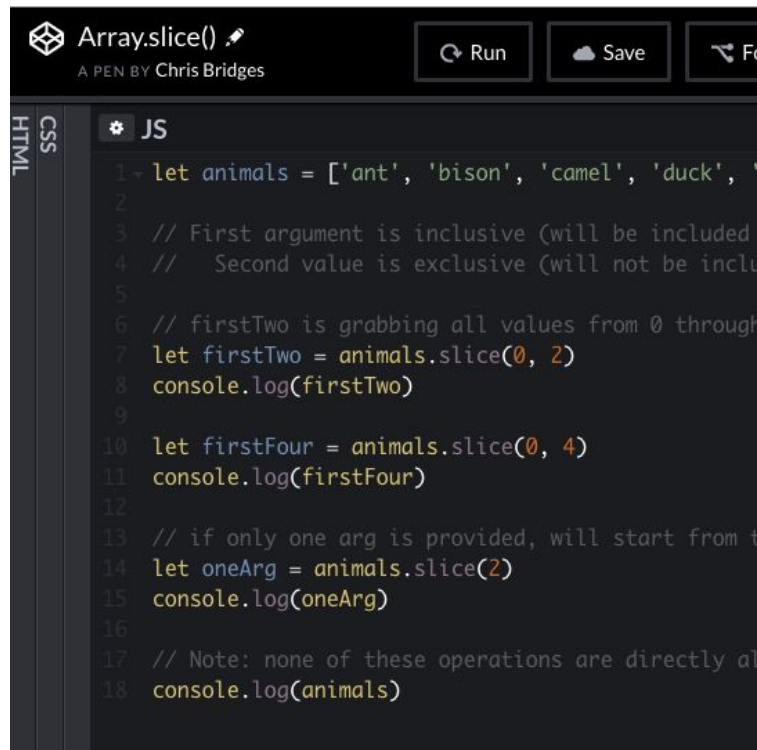
```
let twoThroughFour = myArr.slice(1, 4)
```

```
// [2, 3, 4]
```

```
let threeAndBeyond = myArr.slice(2)
```

```
// [3, 4, 5]
```

Array.slice()



The screenshot shows a code editor with a dark theme. At the top, there's a title bar with a logo, the text "Array.slice()", a small edit icon, and buttons for "Run", "Save", and "Find". Below the title bar, on the left, are tabs for "HTML", "CSS", and "JS", with "JS" being the active tab. The main area contains JavaScript code with line numbers 1 through 18. The code defines an array of animals and demonstrates the slice method with various arguments and comments explaining its behavior.

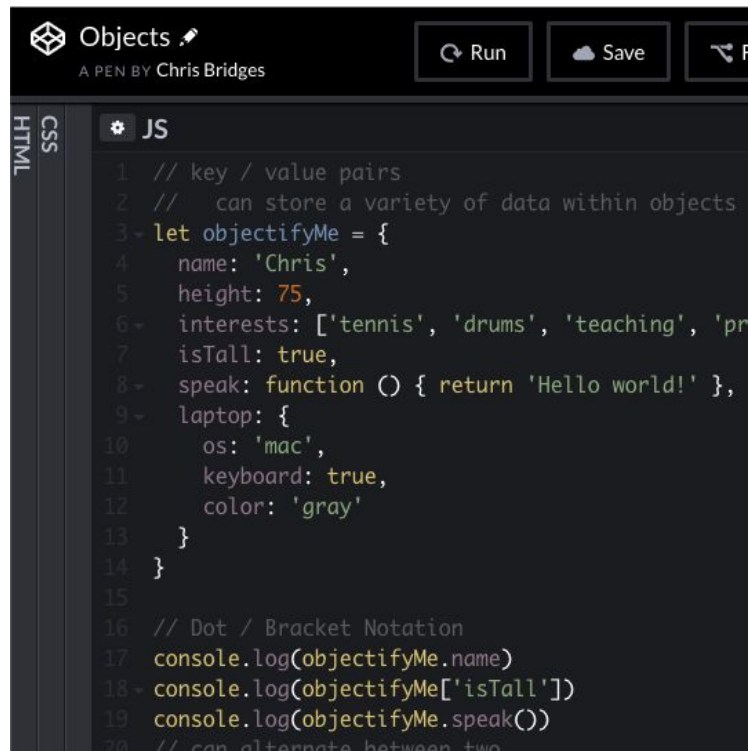
```
1 let animals = ['ant', 'bison', 'camel', 'duck', 'elephant'];
2
3 // First argument is inclusive (will be included)
4 // Second value is exclusive (will not be included)
5
6 // firstTwo is grabbing all values from 0 through 1
7 let firstTwo = animals.slice(0, 2);
8 console.log(firstTwo);
9
10 let firstFour = animals.slice(0, 4);
11 console.log(firstFour);
12
13 // if only one arg is provided, will start from the beginning
14 let oneArg = animals.slice(2);
15 console.log(oneArg);
16
17 // Note: none of these operations are directly altering the original array
18 console.log(animals);
```

Objects

- A data structure that revolves around key / value pairs
- Can access values using dot or bracket notation

```
let myObj = {  
  
    name: 'Dan',  
  
    isTall: true  
  
}  
  
myObj.a // 1  
  
myObj['isTall'] // true
```

Objects Drills



The screenshot shows a code editor window titled "Objects" by Chris Bridges. The editor has tabs for HTML, CSS, and JS. The JS tab is active, showing a JavaScript object named `objectifyMe` with various properties including a nested object for a laptop. The code is as follows:

```
1 // key / value pairs
2 // can store a variety of data within objects
3 let objectifyMe = {
4   name: 'Chris',
5   height: 75,
6   interests: ['tennis', 'drums', 'teaching', 'pr
7   isTall: true,
8   speak: function () { return 'Hello world!' },
9   laptop: {
10    os: 'mac',
11    keyboard: true,
12    color: 'gray'
13  }
14 }
15
16 // Dot / Bracket Notation
17 console.log(objectifyMe.name)
18 console.log(objectifyMe['isTall'])
19 console.log(objectifyMe.speak())
20 // can alternate between two
```


Object.keys()

- Returns an array of all keys from object

```
let myObj = {
```

```
  a: 1,
```

```
  b: 2,
```

```
  c: 3
```

```
}
```

```
let keys = Object.keys(myObj)
```

```
// ['a', 'b', 'c']
```

Object.keys() Drills

A screenshot of a code editor interface. At the top, there's a title bar with a logo, the text "Object.keys()", and a small edit icon. Below the title bar, it says "A PEN BY Chris Bridges". To the right of the title bar are three buttons: "Run", "Save", and "F". On the left side of the editor, there's a sidebar with three tabs: "HTML", "CSS", and "JS". The "JS" tab is selected. The main area of the editor contains the following JavaScript code:

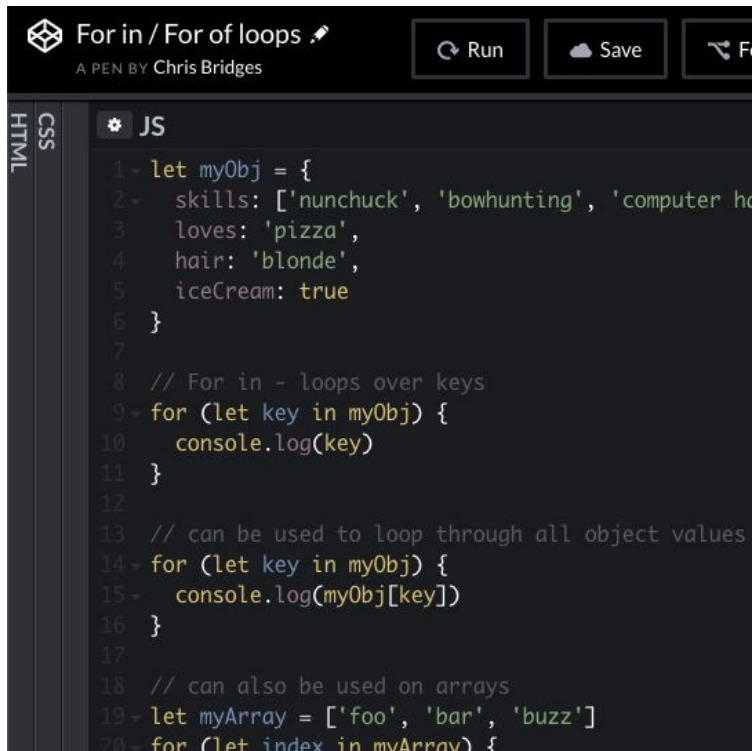
```
1 // returns an array of the object's keys
2
3 let myObj = {
4   skills: ['nunchuck', 'bowhunting', 'computer ha
5   loves: 'pizza',
6   hair: 'blonde',
7   iceCream: true
8 }
9 console.log(Object.keys(myObj))
```

For in / For of Loops

- For in
 - Used to loop over all object keys
- For of
 - Used to loop over all array values

```
let myObj = {  
  a: 1,  
  b: 2,  
  c: 3  
}  
  
for (let key in myObj) {  
  console.log(key)  
}  
  
// a, b, c
```

For in / For of Loops Drills



The screenshot shows a code editor window titled "For in / For of loops" by Chris Bridges. The editor has tabs for HTML, CSS, and JS. The JS tab is active, displaying the following code:

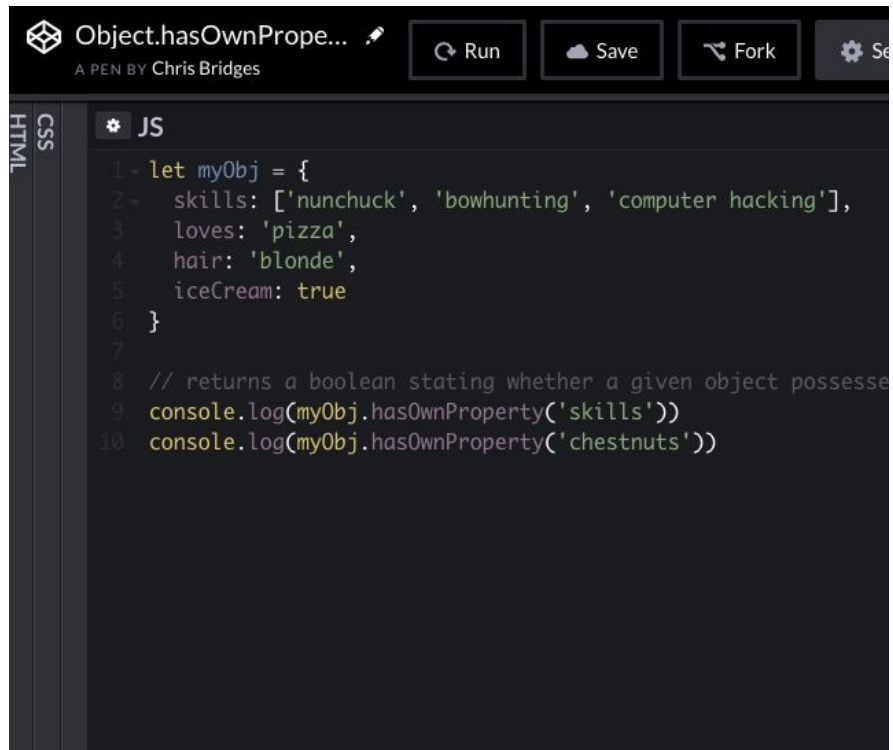
```
1- let myObj = {
2-   skills: ['nunchuck', 'bowhunting', 'computer h
3-   loves: 'pizza',
4-   hair: 'blonde',
5-   iceCream: true
6- }
7
8 // For in - loops over keys
9- for (let key in myObj) {
10   console.log(key)
11 }
12
13 // can be used to loop through all object values
14- for (let key in myObj) {
15-   console.log(myObj[key])
16 }
17
18 // can also be used on arrays
19- let myArray = ['foo', 'bar', 'buzz']
20- for (let index in myArray) {
```

Object.hasOwnProperty()

- Returns a boolean stating whether the given object possesses a particular property / key

```
let myObj = {  
  a: 1,  
  b: 2,  
  c: 3  
}  
  
myObj.hasOwnProperty('a')  
  
// true
```

Object.hasOwnProperty() Drills



The screenshot shows a code editor interface with a dark theme. At the top, there's a header bar with a logo, the title "Object.hasOwnProperty...", a small edit icon, and buttons for "Run", "Save", "Fork", and "Settings". Below the header, the editor is divided into two panes. The left pane has a sidebar with "HTML" and "CSS" tabs, and the "CSS" tab is selected. The right pane is labeled "JS" and contains the following code:

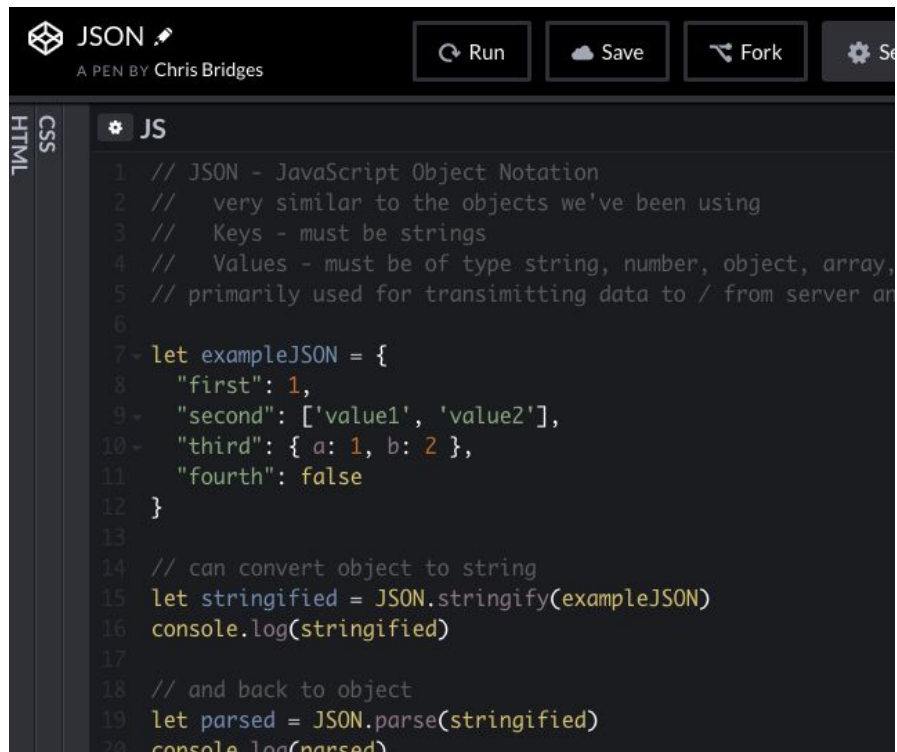
```
1- let myObj = {  
2-   skills: ['nunchuck', 'bowhunting', 'computer hacking'],  
3-   loves: 'pizza',  
4-   hair: 'blonde',  
5-   iceCream: true  
6- }  
7-  
8- // returns a boolean stating whether a given object possesses  
9- console.log(myObj.hasOwnProperty('skills'))  
10 console.log(myObj.hasOwnProperty('chestnuts'))
```

JSON

- Stands for JavaScript Object Notation
- A way of structuring data primarily for the purpose of transmitting data between client / server
- Very similar to the objects we have been working with

```
let exampleJSON = {  
    "name": 'Chris',  
    "isTall": true  
}
```

JSON Drills



The image shows a code editor interface for a project titled "JSON" by Chris Bridges. The editor has a dark theme and a sidebar on the left with tabs for HTML, CSS, and JS. The JS tab is active, showing a JavaScript script that demonstrates JSON operations. The script includes comments explaining JSON, defines an example JSON object, and uses the JSON.stringify and JSON.parse methods to convert between JSON strings and objects.

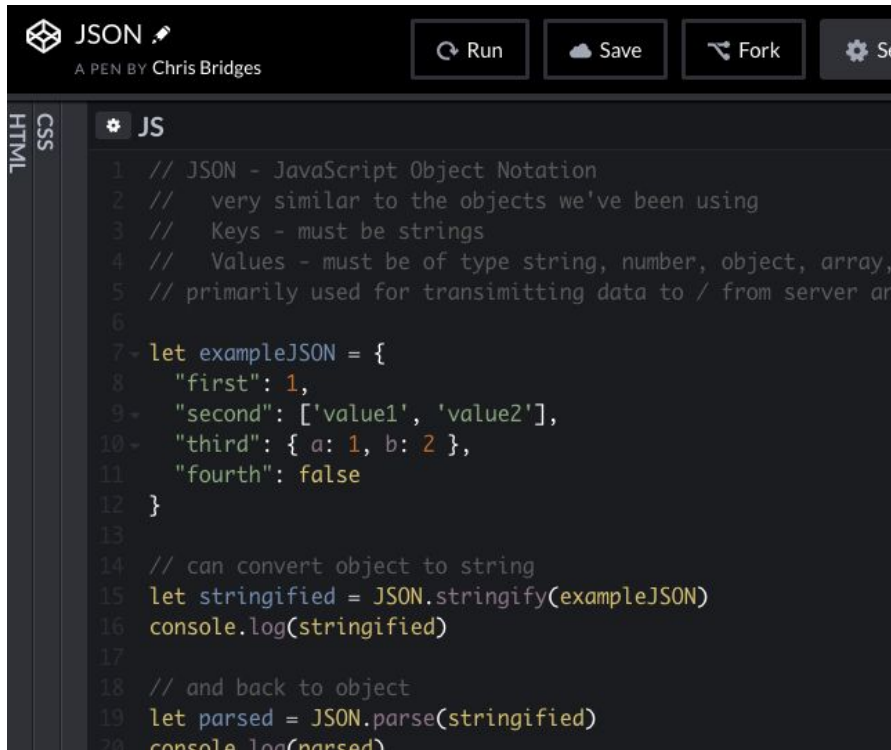
```
1 // JSON - JavaScript Object Notation
2 // very similar to the objects we've been using
3 // Keys - must be strings
4 // Values - must be of type string, number, object, array,
5 // primarily used for transmitting data to / from server and
6
7 let exampleJSON = {
8   "first": 1,
9   "second": ['value1', 'value2'],
10  "third": { a: 1, b: 2 },
11  "fourth": false
12 }
13
14 // can convert object to string
15 let stringified = JSON.stringify(exampleJSON)
16 console.log(stringified)
17
18 // and back to object
19 let parsed = JSON.parse(stringified)
20 console.log(parsed)
```


Summary

- We covered a lot!
 - Data types: Arrays & Objects
 - How to access their data and manipulate it
- There's so much more
 - MDN (Mozilla Developer Network)
 - W3Schools
 - Stack Overflow

Assignments for tonight

bit.ly/tf-git-classroom



The screenshot shows a web-based code editor interface. At the top, there's a header with a logo, the text "JSON", and a subtitle "A PEN BY Chris Bridges". To the right of the header are four buttons: "Run", "Save", "Fork", and "Settings". Below the header, there's a sidebar on the left with tabs for "HTML", "CSS", and "JS". The "JS" tab is selected. The main area displays JavaScript code that demonstrates JSON usage. The code includes comments explaining JSON, an example object, and functions to stringify and parse the object.

```
1 // JSON - JavaScript Object Notation
2 // very similar to the objects we've been using
3 // Keys - must be strings
4 // Values - must be of type string, number, object, array,
5 // primarily used for transmitting data to / from server and
6
7 let exampleJSON = {
8   "first": 1,
9   "second": ['value1', 'value2'],
10  "third": { a: 1, b: 2 },
11  "fourth": false
12 }
13
14 // can convert object to string
15 let stringified = JSON.stringify(exampleJSON)
16 console.log(stringified)
17
18 // and back to object
19 let parsed = JSON.parse(stringified)
20 console.log(parsed)
```



jquery changing background color



jquery changing **css**

jquery changing **text**

jquery changing **background color**

jquery changing **class**



Ways to learn to code

[w3schools.com](https://www.w3schools.com)



meetup



[codecademy](https://www.codecademy.com)

[freeCodeCamp \(🔥\)](https://www.freecodecamp.org)

Two Week Free Trial

bit.ly/tf-free-ff-trail-replacewithrealone

- Free trial of Full Stack Flex online program
- Start with HTML, CSS & JS
- Personal Program Manager
- Unlimited Q&A Sessions
- Student Slack Community