Nested Data Structures / Wrapup

Introduction

Goals for tonight

- A quick comparison of *var*, *let*, and *const*
- Learn about nested data structures
- Compare and contrast types in JavaScript
- Get tips on how-to-learn and how-to-practice

Declaring variables

Different ways in JS

- *var*
- *let*
- *const*

Comparison of Variable Declaration Keywords

Keyword	Can Reassign	Can Redeclare	Can Mutate	Scope Rules
var	√	√	√	function scope
let	√	✗	√	block scope
const	✗	✗	√	block scope

Gotcha with *if* statements and *let* and *const*

If you create variables inside of the *{}* of an *if/else if/else* statement using the *let* keyword, that variable only exists inside the *{}*.

```
if (firstName === "Matt") {
  let funFact = "you love ice cream"
} else {
  let funFact = "you may or may not love ice cream"
}

console.log(funFact); // Uncaught ReferenceError: firstName is not defined
```

To fix this problem, we will declare the variable first, and *then* assign it inside the *if* statement.

```
let funFact;

if (firstName === "Matt") {
  funFact = "you love ice cream";
} else {
  funFact = "you may or may not love ice cream";
}

console.log(funFact); // works well!
```

> **TIP** **This would be a good place for a ternary**
>
> You may know that JavaScript has a "three-way" operator, the ***ternary*** operator. If you're familiar with that, this could be a very good place to use this:
>
> ```
> let funFact = (firstName === "Matt")
>   ? "you love ice cream"
>   : "you may or may not love ice cream";
> ```
>
> (plus, if you wanted to make that variable declared with *const*, now you could do so)

Nested Data Structures

Reviewing Looping

Before we talk about data structures, let's review loops!

What kinds of loops do we have?

Looping over strings / arrays

- `for (initializer; condition; counter)`
- `while (condition)`
- `for...of`

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

```
for (let i = 0; i < arr.length; i++) {
  console.log(arr[i]);
}
```

```
let i = 0;

while (i < arr.length) {
  console.log(arr[i]);
  i++;
}
```

```
for (let val of arr) {
  console.log(val);
}
```

Looping over objects

- `for...in`
- Make sure you use bracket notation to access the keys!

```
let sf = { name: "SF", population: 880_000 };

for (let key in sf) {
  console.log(key);      // print out keys
  console.log(sf[key]); // print out values

  console.log(sf.key);  // what does this print?
}
```

Nested Data Structures

- A data structure is just — a way of structuring data.
- We'll keep things limited to objects and arrays.
- What's is a nested data structure?
  - It just means one data structure inside of another!

Nested Arrays

An array that contains arrays!

```
let maze = [
  [1, 0, 1, 0],
  [1, 1, 0, 1],
  [1, 1, 0, 1],
  [0, 1, 0, 1],
  [1, 1, 1, 1],
];
```

Find top-left with `maze[0][0]` and bottom-left with `maze[4][0]`

Are you really going to be working with data like this?

YES!

- Almost every game / grid / maze that you can make!
- Any structure with rows and columns
- Info About Nested Arrays <https://www.rithmschool.com/courses/intermediate-javascript/javascript-nested-data-structures-arrays>

Using nested loops

- With for/while loops, it's common to use sequential index vars: *i* → *j*, etc
- With *for…of* loops, you shouldn't use index-let names like *i*, *j*
  - Use meaningful names like *row* or *cell*
- Be mindful about your condition
- Use them to loop over nested data structures
- Used if you need info about rest of structure for a value in structure

Challenge: In Matrix?

Given a *matrix* (an array of arrays), search if a given value is present.

```
let matrix = [
  [0, 0, 1],
  [2, 1, 9],
  [0, 0, 0],
];

inMatrix(matrix, 9);  // true
inMatrix(matrix, 4);  // false
```

Solution

```
function inMatrix(matrix, sought) {
  for (let y = 0; y < matrix.length; y++) {
    for (let x = 0; x < matrix[y].length; x++) {
      if (matrix[y][x] === sought) return true;
    }
  }
  return false;
}
```

using for…of loops

```
function inMatrix(matrix, sought) {
  for (let row of matrix) {
    for (let cell of row) {
      if (cell === sought) return true;
    }
  }
  return false;
}
```

using arr.includes()

```
function inMatrix(matrix, sought) {
  for (let row of matrix) {
    if (row.includes(sought)) return true;
  }
  return false;
}
```

Nested Objects

- It's very common to have objects inside of other objects
- It's very common to have objects inside of arrays
- It's also very common to have arrays as values in objects

```
let instructorData = {
  firstName: "Elie",
  siblings: [
    {
      firstName: "David",
      location: "New York",
    },
    {
      firstName: "Haim",
      location: "Seattle",
    },
    {
      firstName: "Tamar",
      location: "New York",
    },
  ],
  isYoungest: true,
  moreData: {
    homeState: "New Jersey",
    hobbies: ["Playing music", "Coding", "Hiking"],
  },
};
```

Are you really going to be working with data like this?

YES!

- GitHub API <https://api.github.com/users/elie/repos>
- Jeopardy Clues <http://jservice.io/api/clues>
- Info About Nested Objects <https://www.rithmschool.com/courses/intermediate-javascript/javascript-nested-data-structures-objects>

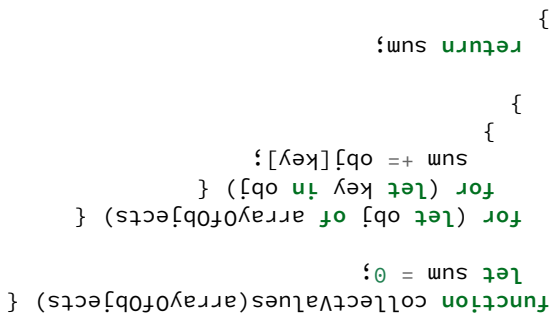
Challenge: collectValues

Given an array of objects where every value is a number, return sum of all values for all objects in the array.

```
let salesTotals = [
  { jan: 10, feb: 25, mar: 15 },
  { jan: 50, dec: 25 },
];

collectValues(salesTotals);  // 125
```

Solution



Types in JavaScript

Primitives

- Primitive types are also known as scalar or simple types.
- Primitive types fit into memory easily.
- JavaScript has five common native primitive types
  - *null*
  - *undefined*
  - *Boolean*
  - *Number*
  - *String*

Reference Types

- A reference type can contain multiple values.
- Reference types are also known as: complex types or container types.
- Reference types in JavaScript include:
  - *Objects*
  - *Array (a kind of object)*
  - *Function (a kind of object)*

Assigning a primitive

When a primitive type is assigned to another variable, a copy of the value of the primitive type is saved in the variable.

Assigning a reference

When a reference type is assigned to another variable, the address of that value is what is copied over.

```
let names = ["Joel", "Alissa", "Nate"];
let namesCopy = names;
namesCopy[2] = "Tim";
names[2] === "Tim"; // true or false?
```

Comparing two primitives

- When comparison operators (`==` and `===`) are used on primitives, they check the type **and** value.

```
let firstName = "Elie";
let firstNameAgain = firstName;
firstName === firstNameAgain;  // true
```

```
firstName === "Elie";  // true
firstNameAgain === "Elie";  // true
```

Comparing two references

- When comparison operators (`==` and `===`) are used on reference types, they check the reference.
- If both refer to the same item, the result is true.

```
let nums = [1,2,3,4];
let nums2 = nums;
nums === nums2;  // true
```

```
nums === [1,2,3,4];  // false
nums2 === [1,2,3,4];  // false
```

Comparing two objects

A quick way:

```
let d1 = {name: "Elie"};
let d2 = {name: "Elie"};

let d1Str = JSON.stringify(d1);
let d2Str = JSON.stringify(d2);

d1Str === d2Str;   // true
```

Another option would be to recursively loop through the objects and make sure each of the properties/values are the same.

How to Learn And Practice

- Repetition is good!
  - To better remember the thing you're reviewing
  - To interest your brain in making new associations
  - Trying to always learn novel things will make you **slower**
- Try things out as you go
  - It's too easy to zone out with videos/books/tutorials
  - Stop, test, question, and play
- Be careful of running code to test it too early
  - Practice *being the computer* — this will help you tremendously!

Learning / Practice Sources

- Find level-appropriate info sources
  - w3schools JS <https://www.w3schools.com/js/default.asp>
  - Rithm School JS Fundamentals <https://www.rithmschool.com/courses#jsfundamentals>
- Find level-appropriate challenges
  - Our Interview Problems <https://www.rithmschool.com/courses/javascript/javascript-rithm-interview-prep>
  - ~6 kyu problems on Codewars
  - Edabit

Bonus Tips

- Find a learning buddy — in person or online
- Keep track of your progress!
  - Inspiration is important and exciting
  - Go back and solve old problems and note what you've learned
- Keep track of any useful code you've written
  - GitHub's gist <https://gist.github.com> is great for this

- Practice the problem solving *process*