Take 2

JavaScript 105

Arrays

Why do we format code?

- When things look similar, they are easier for our brains to process.
- That’s why all books are more or less written the same.
- Y o u d on’t see sentences li k e t h i s v e r y o f t e n.
- When in doubt, use the "Format Document" function from VS Code. There is no such thing as formatting your code too often or too much.

How to format code

- Function calls:

```
// Good
console.log("Hello");

// Bad
console.log( "Hello");
console.log ( "Hello" );
console.log ("Hello");
```

How to format code

- Rule of thumb: if your block has more than one statement, use multiple lines.
- Rule of thumb: only arrow functions with one statement are one-liners

```
// Good
const greet1 = (name) => { console.log("Hi, " + name); };
function greet2(name) {
  console.log("Hi, " + name);
}

// Bad
const greet1 = (name) => { let greeting = "Hi, " + name; console.log(greeting);
function greet2(name) { let greeting = "Hi, " + name; console.log(greeting); }
```

How to format code

- After you start a block or an object with `{`, increase indentation by 2 spaces, starting on the next line. After `}`, reduce indentation by 2 spaces.

```
// Bad
console.log("Test 1");
let obj = {this: 'that'
}
if(3 > 5) {
console.log("What!");
}else {console.log ( "Hello" ) }
```

```
// Good
console.log("Test 1");

let obj = { this: 'that' }

if(3 > 5) {
  console.log("What!");
}
```

Incrementing an integer

- All these commands increment a variable by 1 *if they store a number value*

1. `<identifier> = <identifier> + 1;`
2. `<identifier> += 1;`
3. `<identifier>++;`
4. `++<identifier>;`

```
1  let i = 0;
2  i = i + 1;
3  i += 1;
4  i++;
5  console.log(i); // i is now ... ?
```

How do computers count (again)

- Most humans count with a base of 10 - the *decimal* system.
- Digits are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

- Hexadecimal, digits are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- Decimal 11 in Hex: B

- Binary, digits are: 0, 1
- Decimal 11 in Hex: 1011

- **Computers start 'counting' at 0 not 1**

Arrays

- Sometimes we want to work many values at the same time
- Imagine a list of names in the class. Cumbersome to use them individually.

```
let name1 = "Ari";
console.log(name1);
let name2 = "Bob";
console.log(name2);
let name3 = "Kiri";
console.log(name3);
...
```

Instead, we can do this:

```
let names = ["Ari", "Bob", "Kiri"];
```

- Every value in the array is called an **element**.
- Any value can be an **element**, even `null`.
- Arrays are also objects and have methods
- Can we mix value types in the same array?
- Yes ✅ `let things = ["a string", 5, false, null];`

Array properties and methods

- Most used method: `.length` → prints the number of members in the array
- `.includes(<value>)` → `true` if value is in array, otherwise `false`
- `.find(<function>)` → runs the passed function with every element as an argument. First time the passed function returns true, `find` returns.

```
let names = ["Ari", "Bob", "Kiri"];
console.log(names.length); // Prints '3'
console.log(names.includes("Rob")); // Prints false
console.log(
  names.find((element) => { return element.length == 3 })
); // Prints "Ari"
```

Array indexing

Given the following array:

```
let names = ["Ari", "Bob", "Kiri"];
```

How can we print or change just one element?

```
console.log(names[0]); // Prints 'Ari'
console.log(names[1]); // Prints 'Bob'
console.log(names[2]); // Prints 'Kiri'
```

- The number in brackets (e.g. `[0]`, `[1]`) is called the array index and tells JavaScript which element (from the start) we want to address
- Index *always* starts at 0
- Where have we seen this syntax `[..]` before?

Can we also use dot notation with arrays?

```
let cars = ["Ferrari LaFerrari", "Kia Rio"];
cars.0 = "Holden Astra";
```

No. This produces an error.

Arrays are collections of values

(objects, strings, booleans, mixed, ...)

What can we do with them?

Adding and removing elements:

- Most often used and fast: `push` and `pop`
- To add an element to the **end** of an array, use `push()`:
- To remove and return an element from the **end** of an array, use `pop()`;

- Sometimes useful and slower: `shift` and `unshift`
- To add an element to the **start** of an array, use `unshift()`:
- To remove and return an element from the **start** of an array, use `shift()`;

`push` / `pop` / `shift` / `unshift`

```
let names = [];
names.push("Bob"); // names is now ['Bob'];
names.push("Kiri"); // now ['Bob', 'Kiri'];
names.unshift("Alice"); // now ['Alice', 'Bob', 'Kiri'];

console.log(names.shift());
console.log(names.pop());
console.log(names.pop());
console.log(names.pop());
```

Spotlight: implicit returns

- Why do we need anonymous, named, and arrow functions?

- Arrow functions can have an *implicit return*.
- Without using the `return` keyword, it just returns the result.
- Use `( )` instead of `{ }` to use an implicit return.

```
function longTriple(number) {
  return number * 3;
}

const shortTriple = (number) => (number * 3);
```

- Only works with **one** statement.

```
let shortFunction = (num) => (num = num * 3; num + 2);
```

Output:

```
let shortFunction = (num) => (num = num * 3; num + 2);
                                             ^

Uncaught SyntaxError: Unexpected token ';'
```

Spotlight: evaluation order

- `console.log()` is a function
- `Array.includes()` is a function

```
1  let names = ["Bob", "Dwayne"];
2
3  let doesTheArrayIncludeRob = names.includes("Rob");
4  console.log(doesTheArrayIncludeRob); // Prints false
```

```
// Same as
console.log(names.includes("Rob")); // Prints false
```

```
let ourLittleFunction = (num) => { console.log(num); }

ourLittleFunction;
```

```
let ourLittleFunction = (num) => { console.log(num); }

ourLittleFunction(3);
```

- Some functions expect another function as their argument

```
let names = ["Bob", "Dwayne", "Rico"];
let result = names.find((element) => (element.length == 4));

console.log(result); // Prints "Rico"
```

- Same effect as last slide, but *spelled out with more code*.

```
let finderFunction = function(element) {
  return (element.length == 4);
}
let result;

if(finderFunction("Bob")) {
  result = names.find(finderFunction);
} else if(finderFunction("Dwayne")) {
  result = names.find(finderFunction);
} else if(finderFunction("Rico")) {
  result = names.find(finderFunction);
}

console.log(result); // Prints "Rico"
```