Take 2

JavaScript 103

102 review

Boolean operators

- `&&` - AND
- `||` - OR
- `==` - EQUAL

More boolean operators

- `!` - NOT - turns `true` into `false` and `false` into `true`
- `!true` -> `false`
- `!=` - NOT EQUAL
- `5 == 5` -> `true`
- `5 != 5` -> `false`

- `!` - NOT
- `!=` - DOES NOT EQUAL

```
1  !true = ?
2  !!true = ?
3  !!!true = ?
4  !false || false = ?
5  !true && true = ?
6  true == true
7  true != true
8  true != false
```

Block scope

- Variables declared inside a block do not exist outside the block.

```
let name = "Peter";

{
    let age = 35;
    console.log(age);
}

console.log(name);
console.log(age);
```

Block scope

- Variables can be re-declared in a block without affecting the variable outside the block

```
let name = "Peter";

{
    let name = "John";
    console.log(name);
}

console.log(name);
```

Don’t Repeat Yourself

Instructions (words):

1. Say your name
2. Say your suburb
3. Say your favorite food

Instructions (JavaScript):

```
1  console.log(name);
2  console.log(suburb);
3  console.log(favoriteFood);
```

Don’t Repeat Yourself

Instructions (words):

- Say your name is Andy.
- Say you live in Auckland.
- Say your favorite food is pizza.

- Say your name is Jesiah.
- Say you live in Sydney.
- Say your favorite food is a burger with the works.

Functions

Instructions (Javascript):

```
console.log("My name is Andy");
console.log("I live in Auckland");
console.log("My favorite food is pizza");

console.log("My name is Jesiah");
console.log("I live in Sydney");
console.log("My favorite food " +
"is a burger with the works");
```

Functions

```
1  // One function
2  function introduceYourself(name, suburb, favoriteFood) {
3    console.log("My name is " + name);
4    console.log("I live in " + suburb);
5    console.log("My favorite food is " + favoriteFood);
6  }
7
8  // Two calls
9  introduceYourself("Andy", "Auckland", "pizza");
10 introduceYourself("Jesiah", "Sydney", "a burger " +
11 "with the works");
```

```
// Function definition
function name ( parameters ) { }
// Function call
name ( arguments )
```

Example:

```
// Function definition
function greeting() { console.log("Yo"); }
// Function call
greeting();
```

Example:

```
// Function definition *without* parameters
function greeting() { console.log("Yo"); }

// Function call *without* arguments
greeting();
```

```
// Function definition *with* parameters
function greeting(name) { console.log(`Yo, ${name}`); }

// Function call *with* arguments
greeting("Greg");
```

Example:

```
function greeting(name) {
  console.log(`Yo, ${name}`);
}
greeting("Greg");
```

evaluated as:

```
{
  let name = "Greg";
  console.log(`Yo, ${name}`);
}
```

Exercise

Let’s write some functions together.

Really important: parameter vs argument

```
function introduceYourself(name, suburb, favoriteFood) {
  console.log("My name is " + name);
  console.log("I live in " + suburb);
  console.log("My favorite food is " + favoriteFood);
}

introduceYourself("Andy", "Auckland", "pizza");
```

- `name`, `suburb`, `favoriteFood` are *parameters*.
- When we *call* the function (tell the program to run these instructions), the *parameters* get replaced with the *arguments* from the *call*.

```
1  function introduceYourself(name, suburb, favoriteFood) {
2    console.log("My name is " + name);
3    console.log("I live in " + suburb);
4    console.log("My favorite food is " + favoriteFood);
5  }
6
7  // Function call
8  introduceYourself("Andy", "Auckland", "pizza");
```

- In this example: `name` on L1-5 turns into "Andy".
- In this example: `suburb` on L1-5 turns into "Auckland".
- In this example: `favoriteFood` on L1-5 turns into "pizza".

Define

```
function sayHi(name) {
  console.log(`Hi ${name}`);
}
```

Keyword `function`

Name of the function

`(` list of parameters `)`

`{` instructions `}`

Call

```
sayHi("Andy");
sayHi("Jesiah");
sayHi("Josh");
```

Name of the function

`(` list of arguments `)`

`;`

How to write a function

- Keyword `function`
- Name of the function
- `(` list of parameters `)`
- `{`
- instructions
- `}`

Examples

- Name: 'sayHelloTo', instructions: say hello and then a name
- Name: 'sayGoodbyeTo', instructions: say goodbye and then a name

Let’s see some broken ones

And fix 'em

```
funcion sayHello(name time)
  console log("Good " + time ", " + name)
}

// Should output: 'Good morning, Dave.'
sayHelloTo("morning", Dave)
```

There are other ways to write a function

```
1  // Named function
2  function sayHello(name) { console.log(`Hello ${name}`); }
3
4  // Anonymous function (saved in a variable)
5  const sayHello = function(name) {
6    console.log(`Hello ${name}`);
7  }
8
9  // Arrow function (because of the arrow: =>)
10 const sayHello = (name) => { console.log(`Hello ${name}`); }
```

Mindblowing fact

You've been using a function all along: `console.log`

```
console.log("this is the first argument");
```

Return values

```
function greeting() {
  "Bob";
}

let name = greeting();
// `name` is now undefined
```

`return` statement

```
function greeting() {
  return "Bob";
}

let name = greeting();
// `name` is now "Bob"
```

Use a `return` statement to communicate from the function back to the rest of the program.

Evaluating a line of JavaScript

1. substitute variables
2. run function calls
3. do calculations
4. run line

```
1  function bro(dude) {
2    return dude + 12;
3  }
4  let brodude = 13;
5  console.log(bro(brodude + 1));
```

Recap