W03D2: Javascript Booleans, Control Flow, and Functions

Fullstack Academy

Lesson Objectives

Fullstack Academy

Control Flow

1. `if… else`
    a. Bonus: `?… :`
2. Truthiness and Falsiness
3. Comparisons
4. `if… else if… else`

Boolean Logic

1. The Operators
    a. `&&, ||, !`
2. How to use them

Functions

1. What they are and why we use them
2. How to use them
    a. Declaration: `function, ()=>`
    b. Invocation
    c. Parameters, Arguments
    d. Returns

 |

Fullstack Academy

Tell the computer how to make decisions

Control Flow

 |

Control Flow: `if… else` and the Boolean

Fullstack Academy

”if” statements are one of the most common forms of logic in programming. It is literally a single decision:

```
if (something) {
  console.log("action");
}
```

If something is true, then log ’action’.

If you need to do a different action when something isn’t true, then you can use an if… else statement:

```
if (something) {
  console.log("action");
} else {
  console.log("a different action");
}
```

If something is true, then log ’action’; otherwise, log ’a different action’.

Control Flow: `if… else` nesting

Fullstack Academy

”if” statements can also be nested, for more complex decisions:

```
if (iAmNotHungry) {
  console.log("meh");
} else {
  if (iHaveFood) {
    console.log("nom");
  } else {
    console.log("aww");
  }
}
```

It’s important not to nest too deeply. It makes your logic hard to follow, and changing or maintaining your code very difficult.

A good general rule of thumb is to try to keep nesting to two levels deep at the most!

Control Flow (Bonus): the Ternary operator

Fullstack Academy

There is a shorthand way to write if... else statements.
This is called the ternary, or conditional, operator.

This:

```
if (something) {
  console.log("action");
} else {
  console.log("a different action");
}
```

Can be written as:

```
something ? console.log("action") : console.log("a different action");
```

This is relatively common, but important not to overuse it. Especially avoid nesting using ternaries; things can get weird pretty fast.

Resource:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Conditional_Operator

Control Flow: Truthiness & Falsiness

Fullstack Academy

Javascript considers some values "truthy," meaning that they are regarded as TRUE for the purposes of control flow; and others are "falsy," meaning that they are regarded as FALSE.

Truthy values:

- `true`
- Any non-zero number (including negatives ones!)
- Any non-empty string (including just a space!)

Falsy values:

- `false`
- `null`
- `undefined`
- `0` (the number zero)
- `NaN` (not-a-number)
- `""` (an empty string)

Resources:
https://developer.mozilla.org/en-US/docs/Glossary/Truthy
https://developer.mozilla.org/en-US/docs/Glossary/Falsy

Control Flow: Comparisons

Fullstack Academy

Comparison operators allow you to check whether two things in Javascript are equivalent, different, or greater/less than each other.

Equivalency:

- `==` : Loose equivalence
  - `'2' == 2 // true`
- `===` : Strict equivalence
  - `'2' === 2 // false`
- `!=` : Loose non-equivalence
  - `'2' != 2 // false`
- `!==`: Strict non-equivalence
  - `'2' !== 2 // true`

Greater or Less Than

- `>` : Greater Than
  - `'3'> 2 // true`
- `>=` : Greater Than or Equal To
  - `'2' >= 2 // true`
- `<` : Less Than
  - `'1' < 2 // true`
- `<=` : Less Than or Equal To
  - `'2' <= 2 // true`

 |

Resource: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Expressions_and_Operators#comparison_operators

Control Flow: `if… else if… else`

Fullstack Academy

You can avoid some nesting by using `if… else if… else` statements.

So this:

```
if (iAmNotHungry) {
  console.log("meh");
} else {
  if (iHaveFood) {
    console.log("nom");
  } else {
    console.log("aww");
  }
}
```

Can become this:

```
if (iAmNotHungry) {
  console.log("meh");
} else if (iHaveFood) {
  console.log("nom");
} else {
  console.log("aww");
}
```


DEMO: CONTROL FLOW

Fullstack Academy

BREAK TIME

Truthy-Falsy

1. `0`
2. `"false"`
3. `-1`
4. `"true"`
5. `null`
6. `let isThisTruthyOrFalsy;`
7. `"" + ""`

TRUTHY

FALSY

Control Flow: What will be logged?

```
let myVar;
if (myVar) {
  console.log("hello");
} else {
  console.log("goodbye");
}
```

TYPE YOUR ANSWER:

Control Flow: What will be logged?

```
let iAmHungry = true;
let iHaveFood = false;

if (iAmHungry) {
  if (iHaveFood) {
    console.log('yum');
  } else {
    console.log('oh no');
  }
} else {
  console.log('meh');
}
```

TYPE YOUR ANSWER:

Control Flow: What will be logged?

```
let finishedWork = false;
let iAmKindaTired = true;
let iAmOverThisForReal;

if (finishedWork) {
  if (iAmKindaTired) {
    console.log('go to bed');
  }
} else if (iAmOverThisForReal) {
  console.log('self care');
} else {
  console.log('finish my work');
}
```

TYPE YOUR ANSWER:

Control Flow: What will be logged?

```
let chosen = 1;
console.log(chosen ? 'wee' : 'oo');
```

TYPE YOUR ANSWER:

AND, OR, NOT

Boolean Logic

Boolean Logic

Fullstack Academy

Allows you to represent more complex decision making processes by combining multiple conditions.

Operators:

- `&&` : logical AND
  - EVERYTHING joined by `&&` must be truthy in order for the statement to evaluate as true
- `||` : logical OR
  - AT LEAST ONE of everything joined by `||` must be truthy in order for the statement to evaluate as true
- `!` : logical NOT
  - Anything falsy preceded by `!` will evaluate as true; anything truthy will evaluate as false.

Resource: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Expressions_and_Operators#logical_operators

DEMO: BOOLEAN LOGIC

 |

True or False?

1. `true && false`
2. `true || false`
3. `false || true`
4. `!0`
5. `-1 || (0 && false)`
6. `!!("" + 0)`
7.
```
(1 || 0) &&
( (1 && 0) || "false" )
```

TRUE

FALSE

Fullstack Academy

BREAK TIME

Fullstack Academy

Easily reusable code blocks

Functions

Javascript Functions

Fullstack Academy

Allow you to reuse code over and over again, by assigning a block of code to a variable.

1. Declaration
   a. You create a function by declaring it.
2. Invocation
   a. You use a function by invoking it (aka calling, running, or executing it).
3. Parameters
   a. These are variables that may be initialized when the function is declared
4. Arguments
   a. These are values that may be given to the function, and are assigned to the parameters.
5. Returning values
   a. Most functions will return a value

Resource: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions

Javascript Functions: Declaration

Fullstack Academy

The traditional way to define a function is using the ”function” keyword. This is the one you should lean on the most, for now:

Traditional named function declaration

```
function myFunction() {
  console.log('my function');
}
```

- Creates a function named "myFunction"
- When in doubt, use this

Traditional function expression

```
const myFunction = function() {
  console.log('my function');
}
```

- Creates an unnamed function
- Assigns it to the variable "myFunction"

Resources:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/function

Javascript Functions, Bonus: Arrow functions

Fullstack Academy

A newer way to define a function creates ”arrow functions.” These look cool and are more concise, but are limited in some ways. You don’t need to worry about these yet, but you’ll probably start encountering a lot of them fairly soon.

Arrow function expression

```
const myFunction = () => {
  console.log('my function');
}
```

```
const myFunction = () => console.log('my function');
```

Resource: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

Javascript Functions, Bonus: Anonymous Functions

Fullstack Academy

Sometimes you need a function, but you don't need to give it a name. Anonymous functions are perfect for this! More stuff you don't really need right now, but will encounter soon.

```
// Anonymous traditional function
(function() {
  console.log('my function');
});

// Anonymous arrow function
() => console.log('my function');
```


Javascript Functions, Bonus: IIFEs

Fullstack Academy

Immediately Invoked Function Expressions are just anonymous functions that call themselves. There are various uses for these, but we don't need to worry about them right now. Just know that you'll eventually encounter these.

```
// Traditional IIFE
(function() {
  console.log('my function');
})()

// Arrow IIFE
(() ⇒ console.log('my function'))();
```

Resource: https://developer.mozilla.org/en-US/docs/Glossary/IIFE

Javascript Functions: Invocation

Fullstack Academy

You invoke, or call, a function by using its name, immediately followed by a set of parentheses:

```
function myFunction() {
  console.log('my function');
}

myFunction(); // will log "my function"
```

Resource: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions#calling_functions

Javascript Functions: Parameters

Fullstack Academy

When you declare a function, you can also define parameters. These are just variables you can initialize, to allow passing values into them when calling the function.

```
function myFunction(param1, param2) {
  console.log('my function');
  console.log(param1);
  console.log(param2);
}
```

Resource: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions#function_parameters

Javascript Functions: Arguments

Fullstack Academy

When calling a function, if you pass in values for the parameters, those values are called arguments.

```
function myFunction(param1, param2) {
  console.log('my function');
  console.log(param1);
  console.log(param2);
}

myFunction("argument1", "argument2");
```

Resource: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions#function_parameters

Javascript Functions: Returning Values

Fullstack Academy

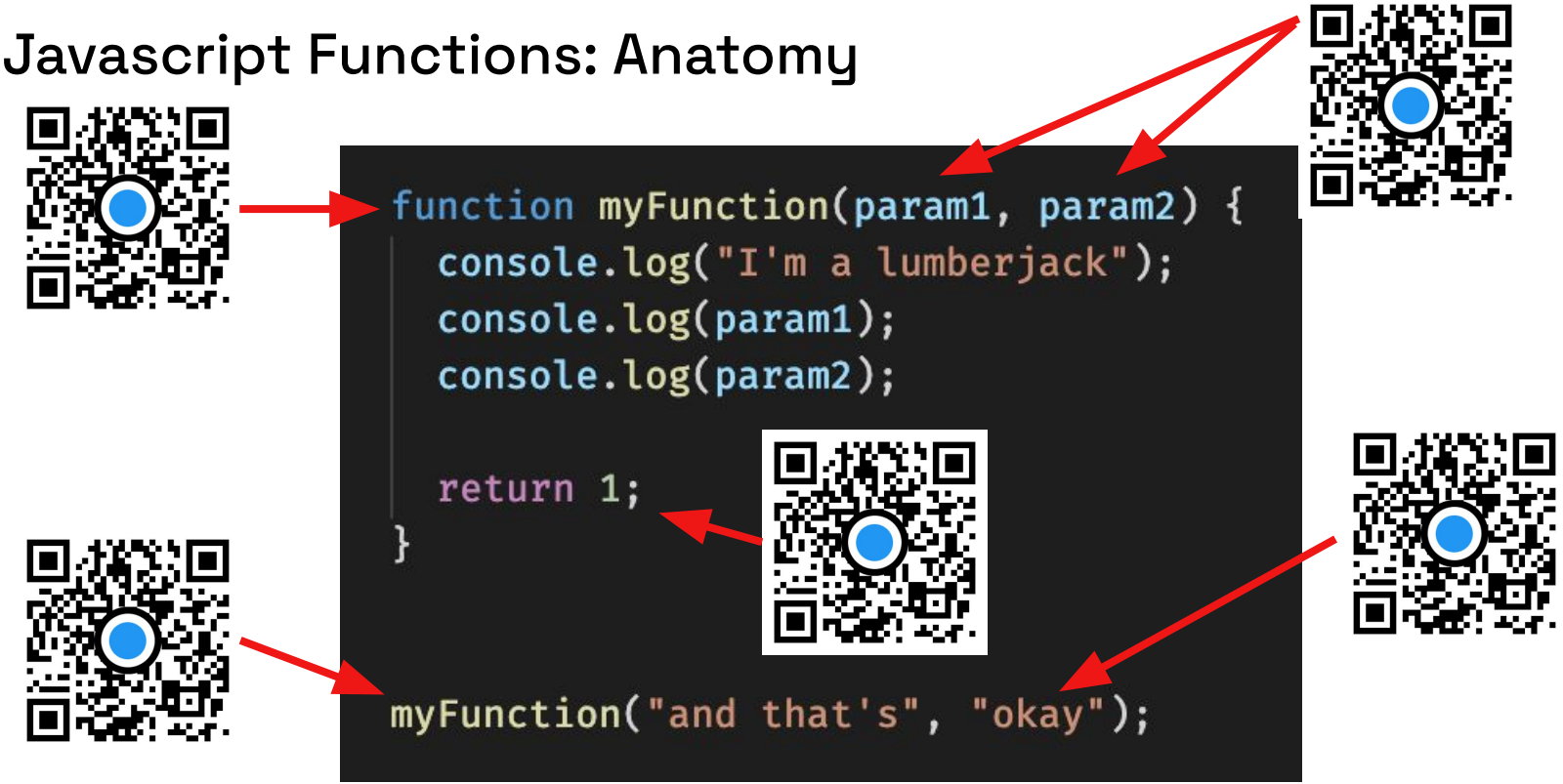
Every function returns a value.
If you don't define a return, the function will return `undefined`

```
function myFunction() {
  console.log("I return undefined");
}

function willReturnSomething() {
  console.log("I return something");
  return "something";
}
```

Resource: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/return

Javascript Functions: Anatomy



DEMO: FUNCTIONS

