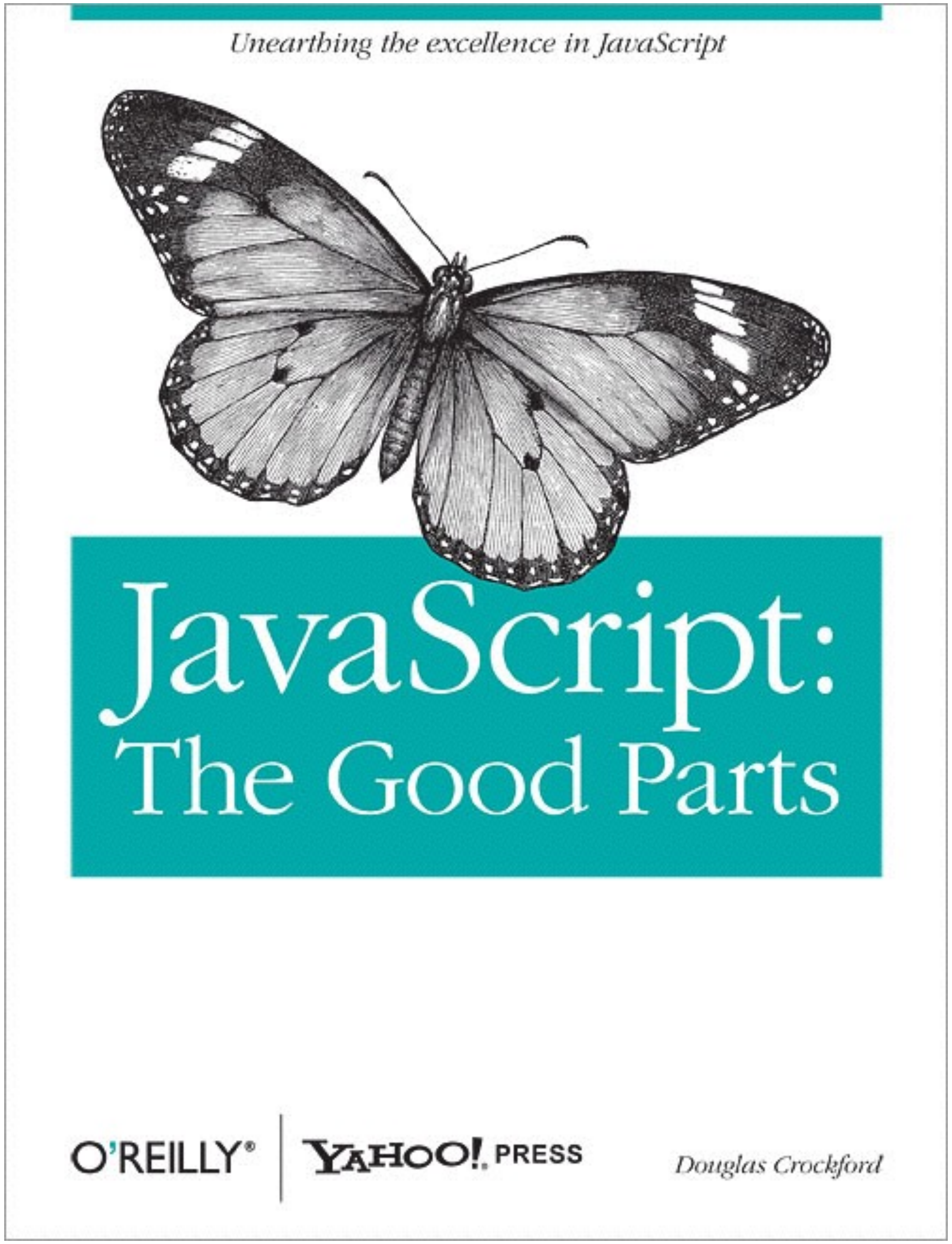
Lesson 3

Conditionals and (More) Loops



Unearthing the excellence in JavaScript

JavaScript: The Good Parts

O'REILLY® | YAHOO!. PRESS

Douglas Crockford

Last week, on a very special episode of JavaScript Development

- We described the concept of a "data type" and how it relates to variables.
- We declared, assigned to, and manipulated data stored in a variable.
- We created arrays and accessed values in them.
- We iterated over and manipulated values in an array.

Learning Objectives

- Use if/else conditionals and switch/case conditionals to control program flow.
- Use Boolean logic (!, &&, ||) to combine and manipulate conditional tests.
- Differentiate among true, false, 'truth-y', and 'false-y'.
- Review loop iteration using for and forEach, and introduce while loops.

Conditional Statements

- Allow us to decide which blocks of code to execute and which to skip, based on the results of tests that we run.

- JavaScript supports two conditional statements: **if...else** and **switch**.

- We'll start off with the **if...else** statement, which uses Boolean (true or false) tests.

if…else

if

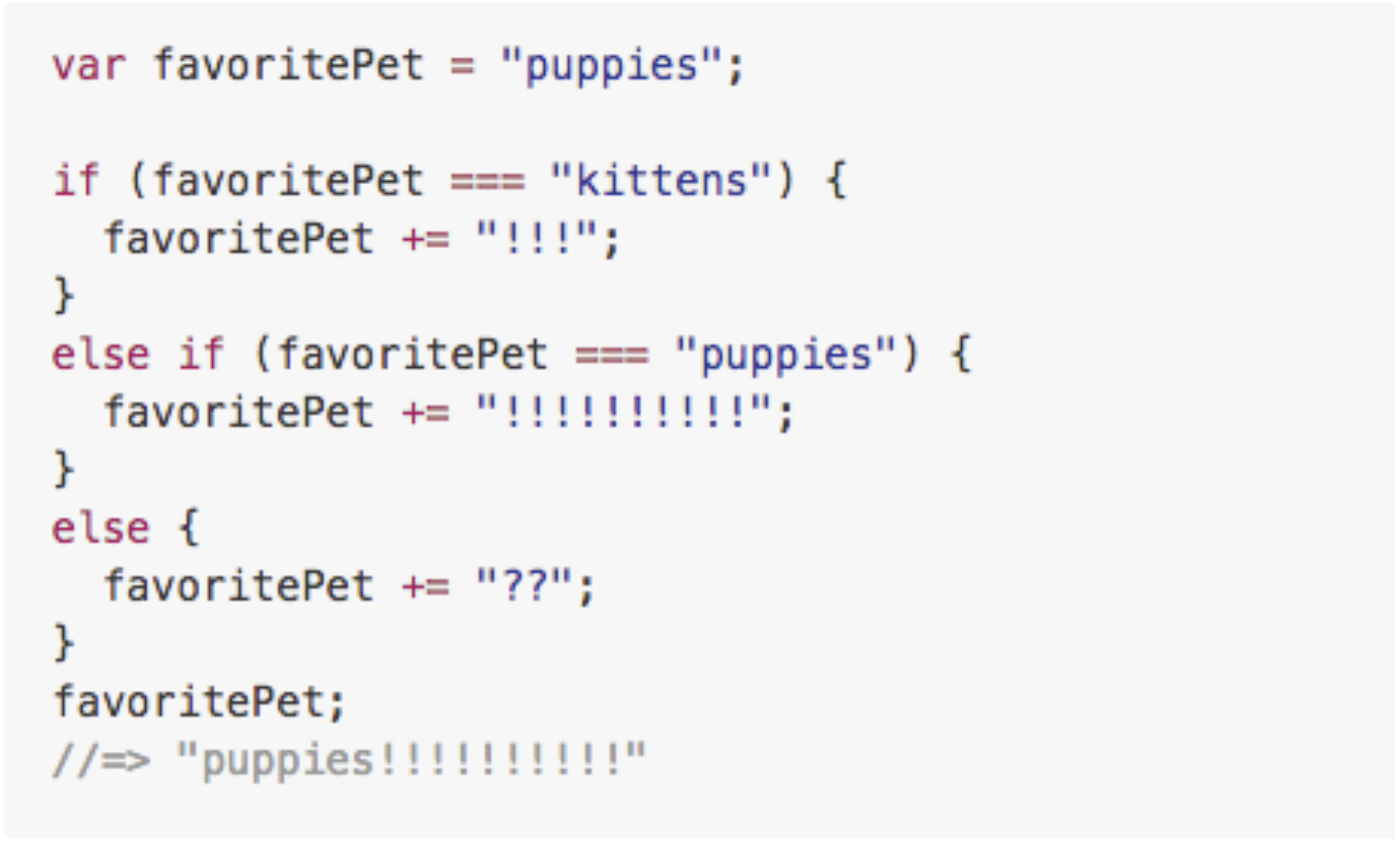
```
if (1 > 0) {
  console.log("hi");
}
//=> hi
```

if…else

```
if (0 > 1) {
  console.log("hi");
}
else {
  console.log("bye");
}
//=> bye
```

if…else if…else

- When you need to test more than one case:



```
var favoritePet = "puppies";

if (favoritePet === "kittens") {
  favoritePet += "!!!";
}
else if (favoritePet === "puppies") {
  favoritePet += "!!!!!!!!!!!";
}
else {
  favoritePet += "??";
}
favoritePet;
//=> "puppies!!!!!!!!!!!"
```

Anatomy of if…else

- Starts with **if**
- Contains a conditional wrapped in parentheses:
  - (4 > 2)
- Contains code wrapped in curly braces, to be executed if the condition is true.
- optional (and similar) **else** clause, to be executed if the condition is false

Ternary operator

- The ternary operator is basically a concise "if-else" in one line,

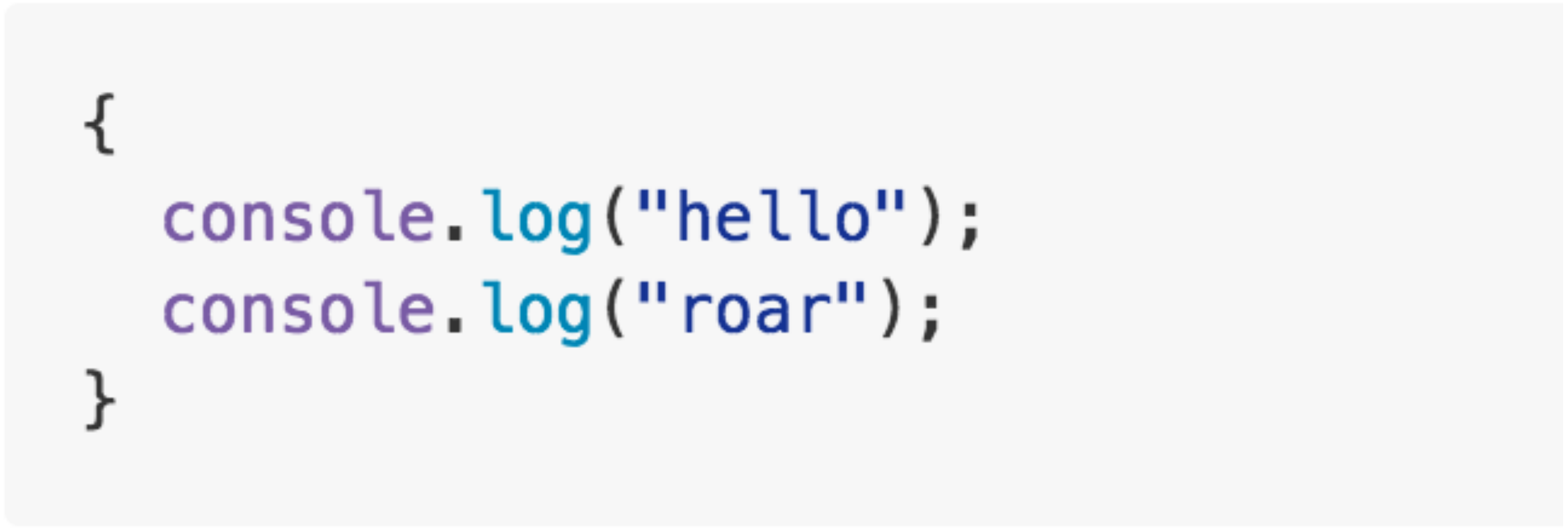
- except that it not only executes blocks of code, it also returns a value:

```
var age = 12;
var allowed = (age > 18) ? "yes" : "no";
allowed;
//=> "no"
```

Block Statements

- Statements intended to be executed after a control flow operation will be grouped into a block statement

- they go inside curly braces



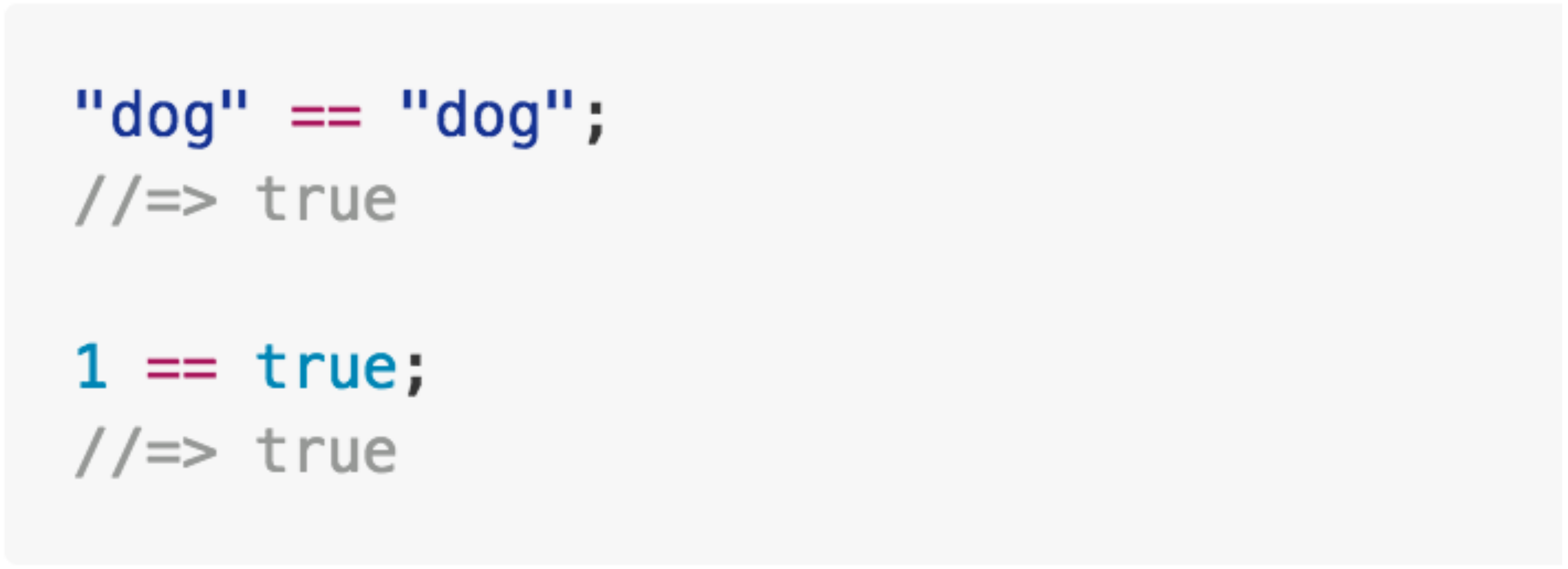
```
{
  console.log("hello");
  console.log("roar");
}
```

Comparison operators

- Comparisons using **<**, **>**, **<=** and **>=**
- **(0 < 10)**
- **(5 <= 5)**
- **(6 > 3)**
- **(7 >= 1)**

Equality operator: ==

- **==** (does type coercion)

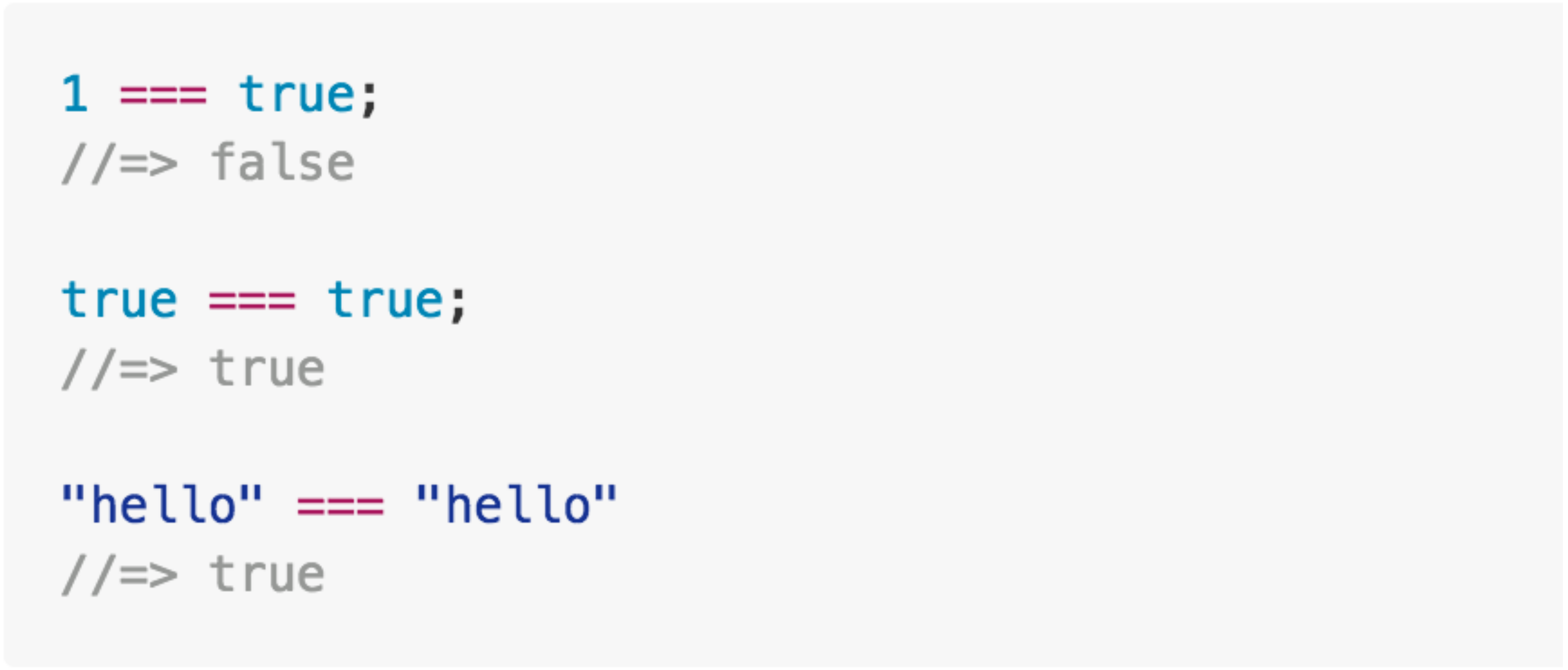


```
"dog" == "dog";
//=> true

1 == true;
//=> true
```

Equality operator: ===

- **===** (does not do type coercion. Use this one!)



```
1 === true;
//=> false

true === true;
//=> true

"hello" === "hello"
//=> true
```

Equality on reference types

- Equality on objects and arrays test whether they are the same thing in memory, not whether they contain the same values:

```
{} === {}
//=> false

[] === []
//=> false

[1,7] === [1,7]
//=> false
```

Equality on reference types (explanation)

The examples in the previous slide equality tests because both object literals and arrays are objects, not just "primitive" values like strings, numbers, and Booleans. Objects and arrays are complex collections of values, and when we refer to them, we're actually referencing where they live in memory. That's why we call them "reference types." Strings and numbers are "value types."

What does this all mean? When we attempt to compare two objects or arrays with ===, JavaScript doesn't care if they look like similar collections. It only compares whether or not they are the exact same object in memory. In each case above, checking for equality is actually comparing two objects that are in two different places in memory. They're not exactly "the same."

Non-equality

- **!=** and **!==** mean “not equal to”
- Use **!==**, not **!=** (because the latter does evil type coercion)

Codealong: comparison operators

- Open your terminal
- Type **node**

Truthy and Falsey

- All of the following become false when converted to a Boolean
  - false
  - 0
  - "" (empty string)
  - NaN
  - null
  - undefined
- All other values become true when converted to a Boolean

Boolean and logical operators

- When you feed Boolean values of true or false into logical operators, they will return true or false based on a few rules.
- There are two "binary" operators that require two values:
  - AND, denoted **&&**
  - OR, denoted **||**
- A third "unary" operator requires only one value:
  - NOT, denoted **!**

Short-circuit logic

- **&&** and **||** and **!** don't have to operate only on true or false -- they can operate on any values, and JavaScript will evaluate the truthyness or falseyness of the operands.
- In the case of **!**, it returns a Boolean true-or-false,
- But in the case of **&&** and **||**, it returns one of the original operands themselves, using short-circuit logic.
- This means that the execution of the second operand is dependent on the execution of the first.

Short-circuit: Making sure something’s not null

```
var name = person && person.name;
```

- In this case, if the first operand person is **undefined**, which is falsey, the second operand **person.name** will not be evaluated. The expression basically says, "We already know the whole **&&** expression is false, because **person** is falsey. Why bother dealing with the second operand?"

Short-circuit: Setting default values

```
var name = person.name || "Bobby Default";
```

- In this case, if the first operand **person.name** turns out to be falsey for any reason (probably because it's **undefined** or it's an empty string), **"Bobby Default"** will be returned. If **person.name** is truthy (probably because it's a non-empty string), it will be returned, and the second operand won't be evaluated. The expression basically says, "We already know the whole **||** expression is true, because **person.name** is truthy. Why bother dealing with the second operand?"

Exercise: What you can do at what age

- Write a program that outputs results based on user's age. This exercise draws on if/else statements, boolean logic, and comparison operators. See the conditions below:

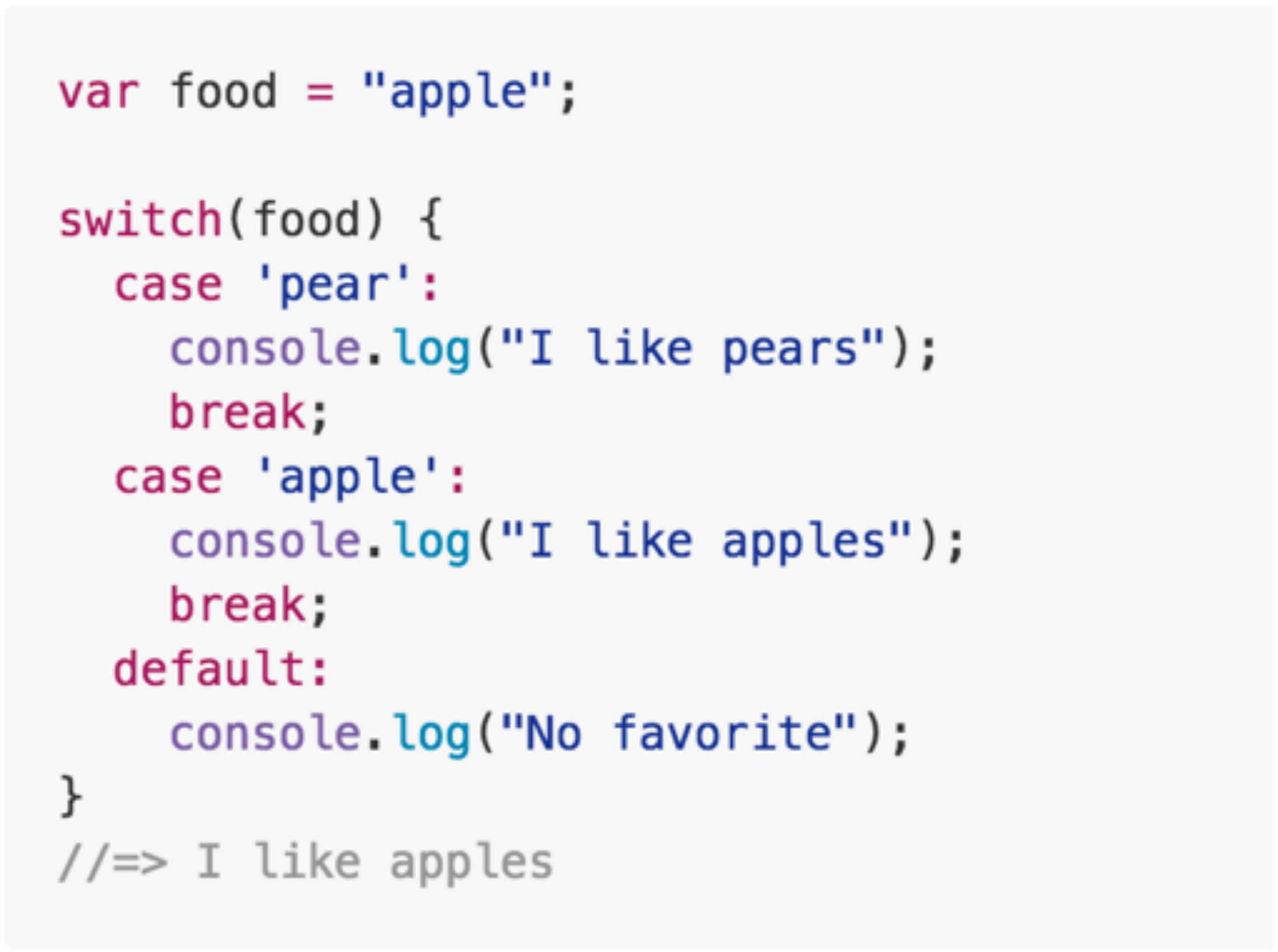
  If you are under 16, you cannot do much...but got to school
  If you are 16 or older, you can drive
  If you 18 or older, you can vote
  If you are 21 or older, you can drink alcohol
  If you are 25 or older, you can rent a car
  If you are 35 or older, you can run for president
  If you are 62 or older, you can receive social security benefits

- Work in pairs, more experienced with less experienced

Break

Switch statements

- These conditional statements can be used for multiple branches based on the value of a number or string.



```
var food = "apple";

switch(food) {
  case 'pear':
    console.log("I like pears");
    break;
  case 'apple':
    console.log("I like apples");
    break;
  default:
    console.log("No favorite");
}
//=> I like apples
```

(The **default** clause is optional)

Codealong:
switch statements

- Open your terminal
- type **node**

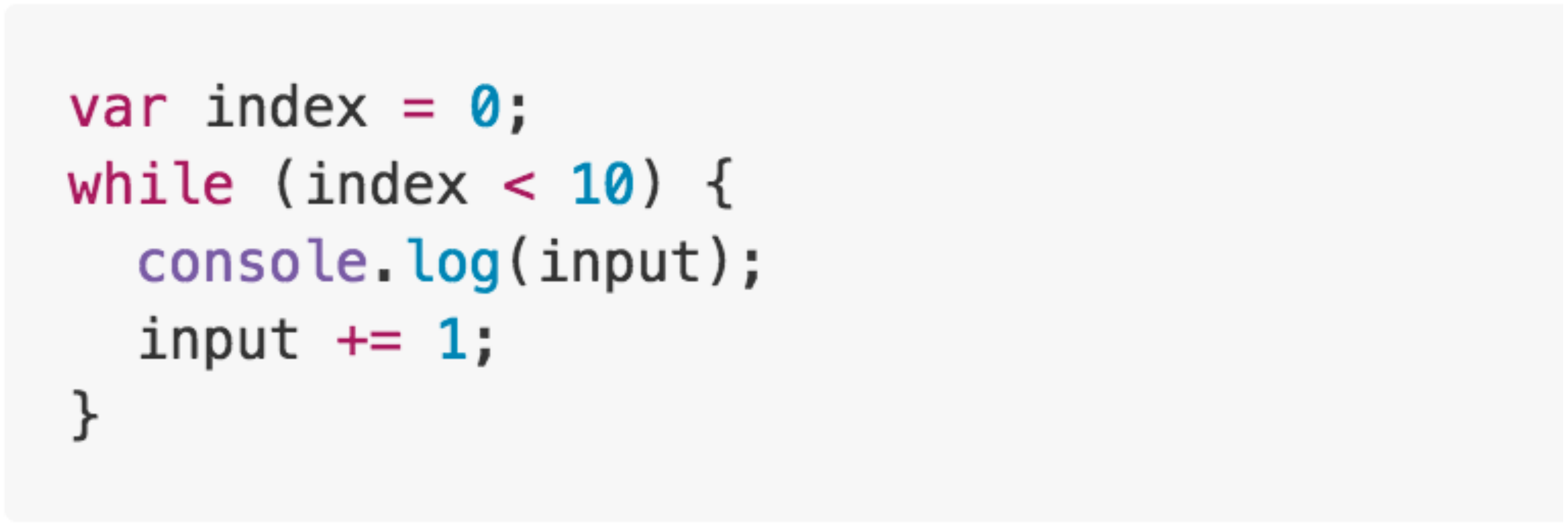
Exercise:
convert if…else to switch

More codealong: switch statements

- Open your terminal
- type **node**

while loops

- **while** is a loop statement that will run “while” a condition is true.



```
var index = 0;
while (index < 10) {
  console.log(input);
  input += 1;
}
```

Codealong: while loops

- Open your terminal
- type **node**

Codealong:
iteration review

- Open your terminal
- type **node**

Exercise: fizzbuzz

- Relying on your newfound knowledge of loops and if/else statements, incrementally build the common fizzbuzz loop. Fizzbuzz is a game about division. Create a program that will iterate through numbers from 1 to 101 and log each number in the console.
- Instructions in the file at http://bit.ly/ga-js2-lesson3-fizzbuzz
- We'll go through it step by step as we go along.

Conclusion

- Be able to explain if/else and switch statements as well as use cases.
- Differentiate between true, false, 'truthy', and 'falsey'.

Homework 1:
99 bottles of beer on the wall

- Make a JavaScript script that prints the lyrics to 99 bottles of beer on the wall. Take into account the singular and plural versions of “beer” and “beers”.

Homework 2: generate random addresses

- Generate random addresses!
- For this I'd like you to:
    - Create new arrays containing dummy data for Street, City, State.
    - Now each time you run the script, it should print new randomly generated address from the arrays and random numbers for street numbers and zip codes.
    - For example:
        - node random-address.js
        - => 34578 Dolphin Street, Wonka NY, 44506
    - As a bonus, randomly include an apartment number (so sometimes an address has an apartment while others do not).