Comparison and Logical Operators and Booleans

The Logical Operators AND and OR

- In addition to NOT, the logical operators OR (||), and AND (&&) give us the ability to control the flow of our programs. But first we have to understand some of their odd quirks.

- && and
- || or
- ! Not

The Logical Operators AND and OR

The boolean operators !, ||, and && follow a set of rules that determine how they behave:

- NOT(!): If the value is *truthy*, return false; if the value is *falsey*, return true.
- OR (||): Return the first *truthy* value; if both values are *falsey*, return the last *falsey* value. OR is nicknamed the "default operator" (can you explain why?)
- AND (&&): Return the first *falsey* value; if both values are *truthy*, return the last *truthy* value.
- AND is nicknamed the "guard operator" (can you explain why?)

Knowledge Check

Let's take a closer look at the "truth table" produced by investigating all possible comparisons of true and false using AND:

```
true && true //=> true
true && false //=> false
false && true //=> false
false && false //=> false
```

Question: In the above table, what is the only case in which AND evalutes to true? You can test the code at the console and check if it matches your guess.

Knowledge Check

Now let's do the same for all possible combinations of true and false using OR:

```
true || true //=> true
true || false //=> true
false || true //=> true
false || false //=> false
```

Beyond true and false

If you think you have a handle on AND and OR, think again. When we move beyond boolean values, things start to get a little strange:

```
1 && 1 //=> 1
1 && 0 //=> 0
0 && 1 //=> 0
0 && 0 //=> 0
```

Is that what you expected?

```
1 || 1 //=> 1
1 || 0 //=> 1
 0 || 1 //=> 1
 0 || 0 //=> 0
```

What's actually happening under the hood?

(HINT: think *truthy*!

AND - The "Guard Operator”

Write the following code in the JS console

```
true && "potato" //=> "potato"
true && true && true && "potato" //=> "potato"
```

What's the "guard operator" guarding? When does AND let you through? When does it stop you?

Test Yourself

Type each command given in this JS Bin Console.

Before you press enter, take a moment to think about what value the console will return.

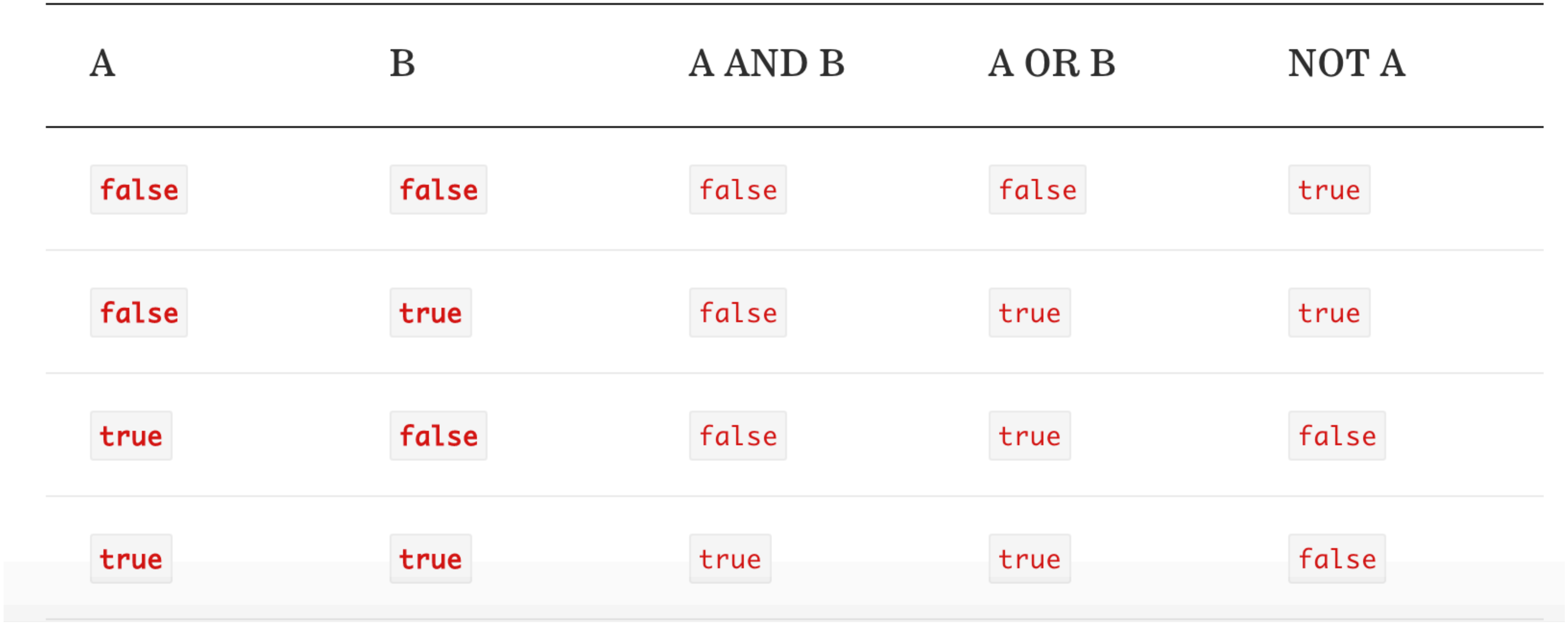
```
1 && "potato”;
false && "potato" 0 && "potato”;
```

OR - The "Default Operator”

```
false || "kiwi" //=> "kiwi"
false || false || false || "kiwi" //=> "kiwi"
```

What's the "default operator" defaulting to? When does it default?

Take a look at the following table, and see if you can predict the results in the last three columns given the values for A and B in each row.



A	B	A AND B	A OR B	NOT A
`false`	`false`	`false`	`false`	`true`
`false`	`true`	`false`	`true`	`true`
`true`	`false`	`false`	`true`	`false`
`true`	`true`	`true`	`true`	`false`

Test Yourself

Type each command given in Console.

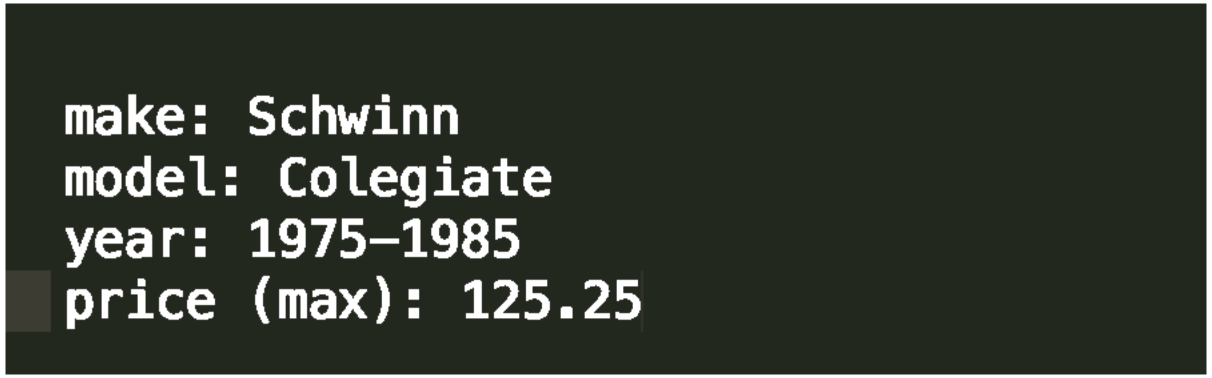
Before you press enter, take a moment to think about what value the console will return.

```
0 || "kiwi"
true || "kiwi"
1 || "kiwi"
1 || true
3 || null
!('')
false && undefined
true && !0
```

All Together Now: Combining Operators and Comparators

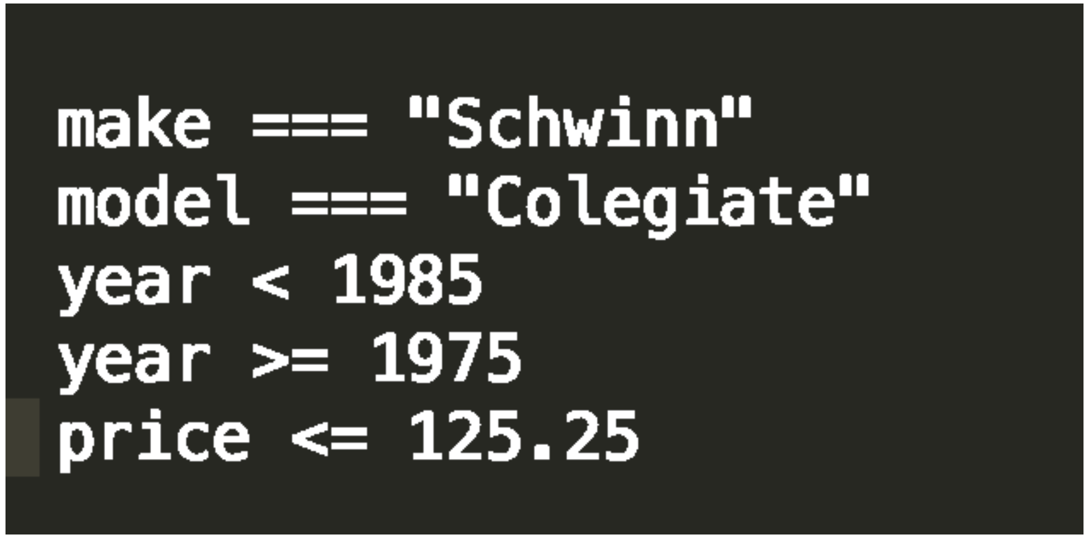
Often we need to combine individual logical statements into larger and larger expressions.

For example, all the following criteria need to be true before I will agree to purchase a vintage bicycle:



```
make: Schwinn
model: Colegiate
year: 1975–1985
price (max): 125.25
```

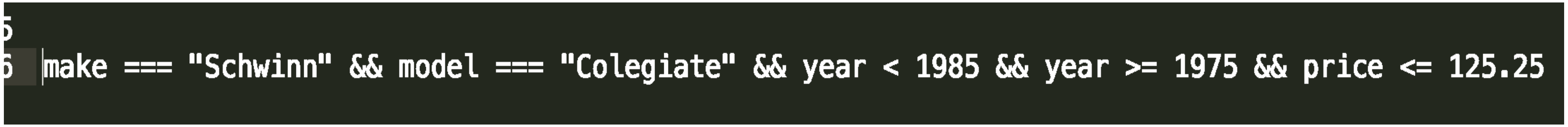
We can translate each of these criterian into code using comparison operators as follows:



```
make === "Schwinn"
model === "Colegiate"
year < 1985
year >= 1975
price <= 125.25
```

All Together Now: Combining Operators and Comparators

Next we can combine them together using our logical operators:



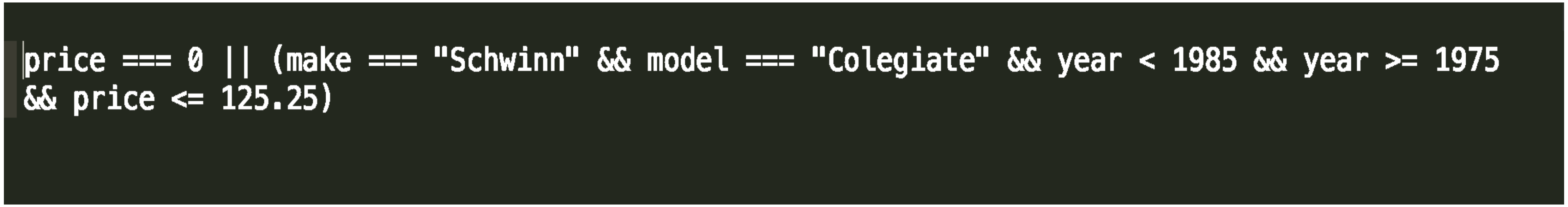
```
make === "Schwinn" && model === "Colegiate" && year < 1985 && year >= 1975 && price <= 125.25
```

But what if I'd also settle for any make/model of bicycle, from any year, if the price was right (which is to say, free)? Is there a way I could incorporate that into my expression in a single line?

There sure is!

All Together Now: Combining Operators and Comparators

Price is the most important factor, so let's address that first!



```
price === 0 || (make === "Schwinn" && model === "Colegiate" && year < 1985 && year >= 1975
&& price <= 125.25)
```

You can definitely squeeze it all into a single line of JavaScript... but we may need some better tools to handle all this complexity!

In an upcoming lesson, we'll take a look at how we can do just that using conditionals.

Exercise

Write the code to perform the actions listed below in the **"JavaScript"** panel in DevTools console or the JS Bin editor.

In the "JavaScript" panel, declare a variable flavor and assign it the value to "chocolate".

Note: After declaring (creating) each variable, be sure to check to make sure you've done things correctly by hitting the "Run" button and then typing the variable name (flavor; in this case) in the "Console" panel and hitting the return/enter key to check its value. You'll want to do this after each step.

Create a variable numberScoops and assign it the value 3.

Create a variable outsideTemperature and assign it the value 75.

Create a variable price and assign it the value 3.5.

Exercise

Create a variable buyIceCream and set it equal to an expression that incorporates the following:

- price is *less than or equal to* 2.5 **OR**
- outsideTemperature is *greater than or equal to* 70 **AND**
- flavor is *equal to* "chocolate" **AND**
- numberScoops is *greater than* 1

After hitting "Run", type buyIceCream; into the console and hit the return/enter key to see whether or not you should buy that delicious chocolate ice cream cone under the given conditions.

Stuck? Try first and try second then ask for Instrutor’s help.