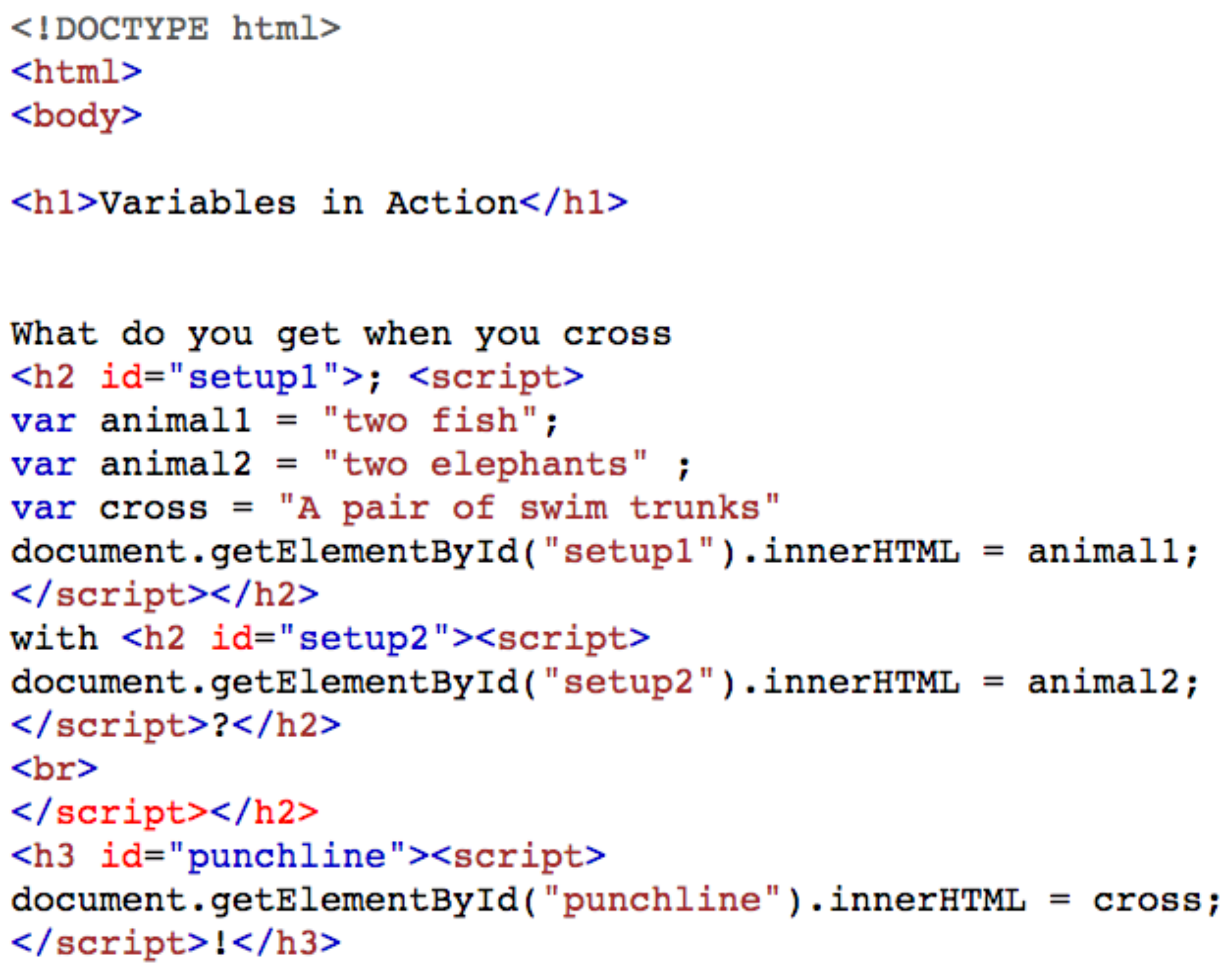
Introduction to Variables

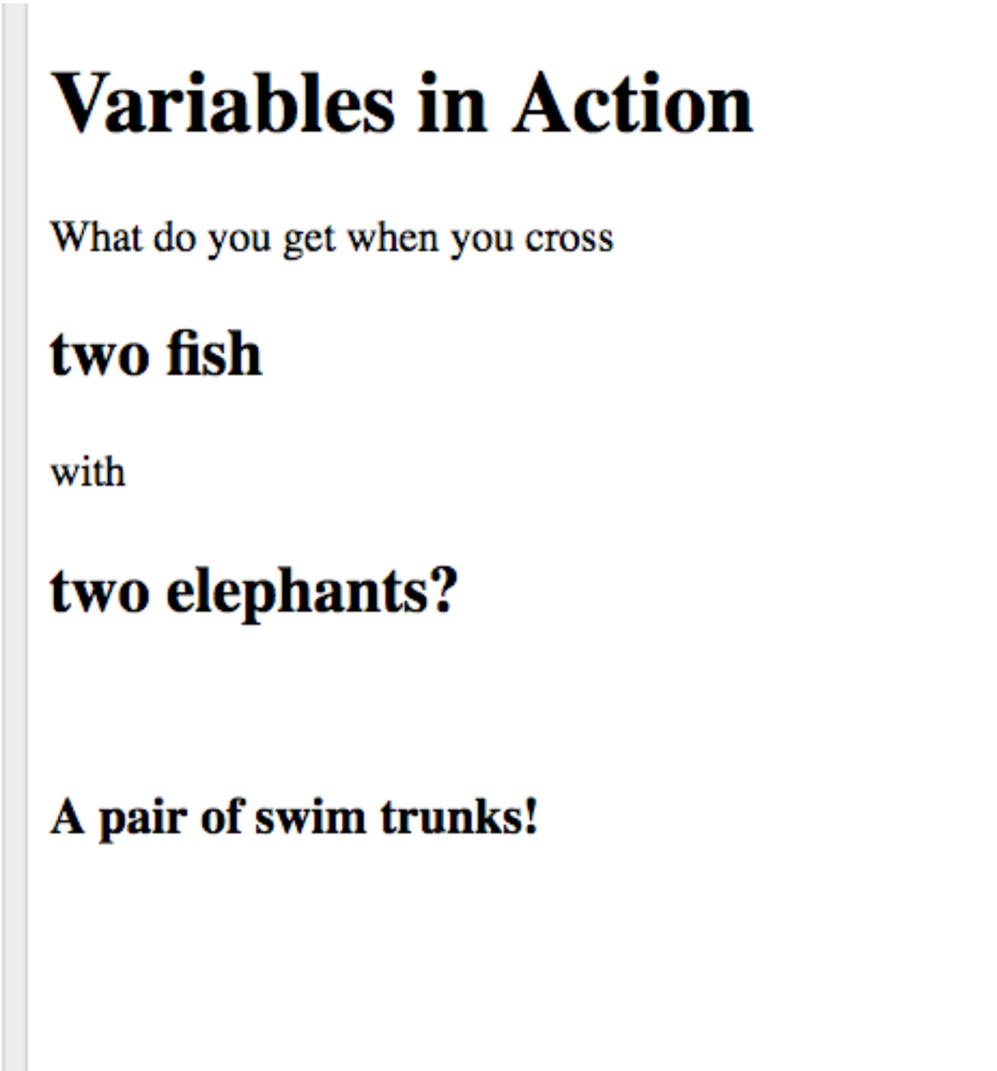
Check out this code (left side) and its rendering (right side):



```
<!DOCTYPE html>
<html>
<body>

<h1>Variables in Action</h1>


What do you get when you cross
<h2 id="setup1">; <script>
var animal1 = "two fish";
var animal2 = "two elephants" ;
var cross = "A pair of swim trunks"
document.getElementById("setup1").innerHTML = animal1;
</script></h2>
with <h2 id="setup2"><script>
document.getElementById("setup2").innerHTML = animal2;
</script>?</h2>
<br>
</script></h2>
<h3 id="punchline"><script>
document.getElementById("punchline").innerHTML = cross;
</script>!</h3>
```



Variables in Action

What do you get when you cross

two fish

with

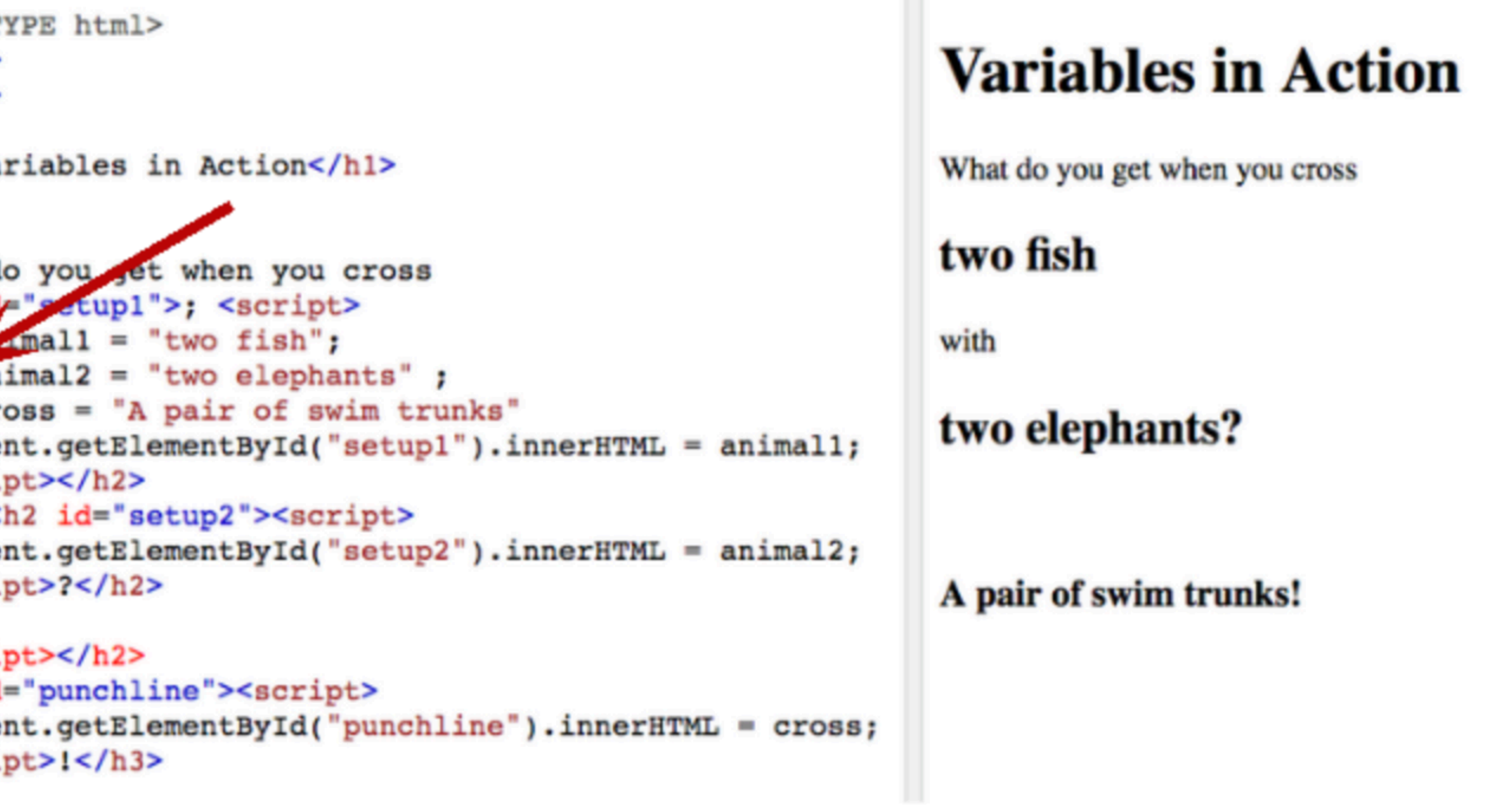
two elephants?

A pair of swim trunks!

Introduction to Variables (Continued)

- See the `var`s in the code? Those are announcements. They're saying: "I hereby name the following variables: 1) **animal1**, 2) **animal2**, and 3) **cross**.

- Later, the code calls upon those variables and they render as the values they were assigned (*two fish, two elephants,* and *a pair of swimtrunks!*)



```
YPE html>



riables in Action</h1>



o you get when you cross
="setup1">; <script>
imal1 = "two fish";
imal2 = "two elephants" ;
oss = "A pair of swim trunks"
nt.getElementById("setup1").innerHTML = animal1;
pt></h2>
h2 id="setup2"><script>
nt.getElementById("setup2").innerHTML = animal2;
pt>?</h2>

pt></h2>
="punchline"><script>
nt.getElementById("punchline").innerHTML = cross;
pt>!</h3>
```

Variables in Action

What do you get when you cross

two fish

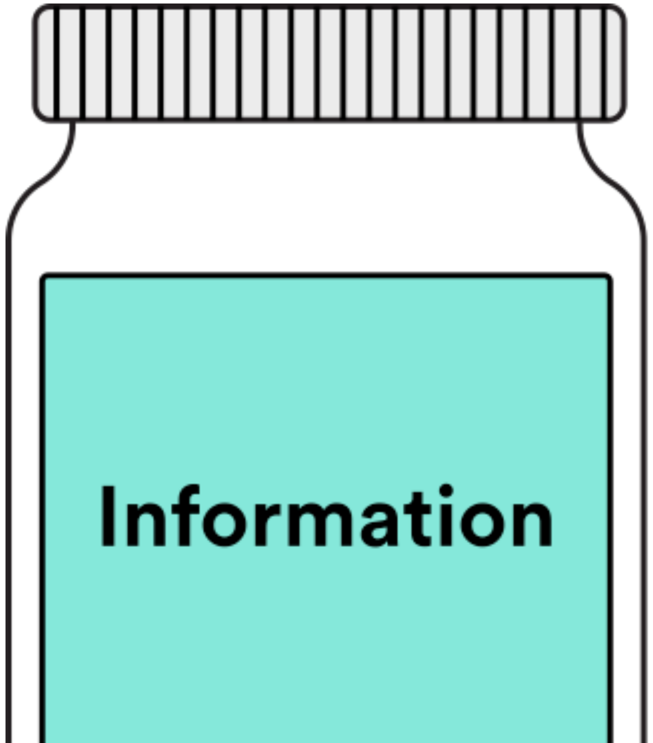
with

two elephants?

A pair of swim trunks!

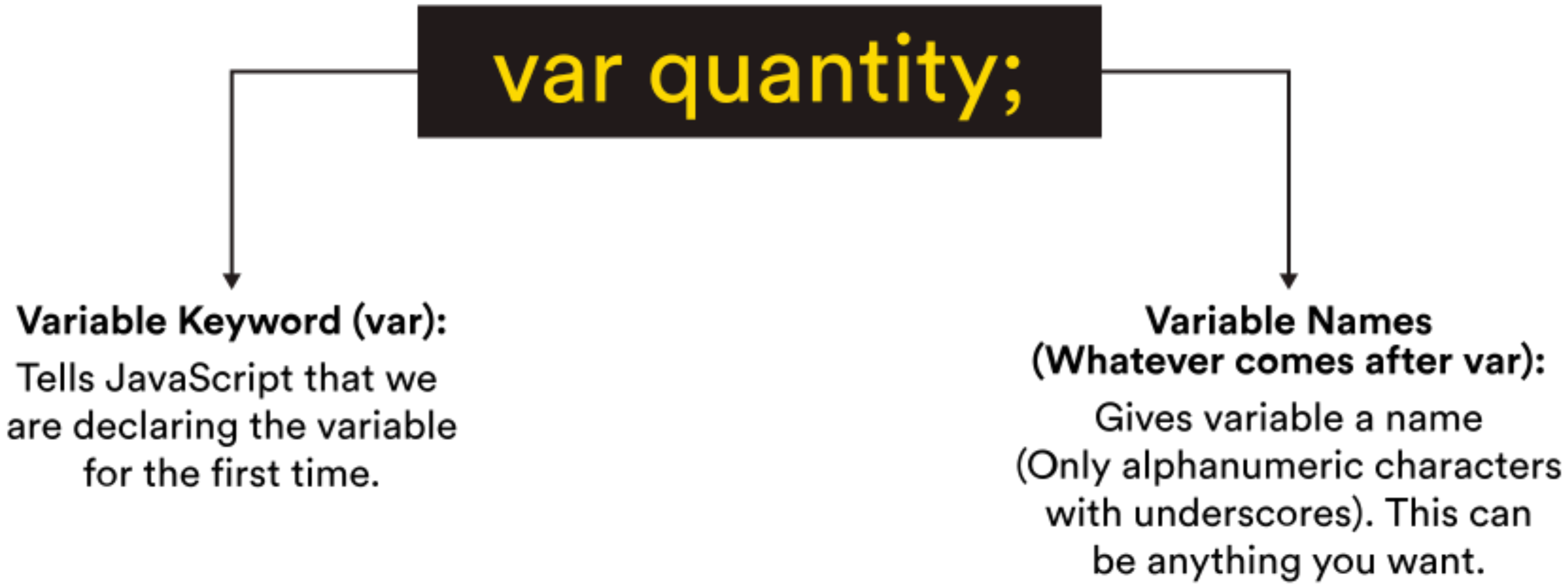
Introduction to Variables (Continued)

- So, what exactly do variables do? Well, they store data types into the memory of the computer so that they can be referenced later.
- Think of them as special containers that can hold information for you.



Announcing Variables

- To use a variable in JavaScript, you simply announce that you want to use it. How? Recall our "swim trunks" example.
- We made the announcement by using the keyword `var` and declaring a variable name next to it.
- **Note:** There must be a space after the keyword var.
-



```
var quantity;
```

Variable Keyword (var):
Tells JavaScript that we are declaring the variable for the first time.

Variable Names (Whatever comes after var):
Gives variable a name (Only alphanumeric characters with underscores). This can be anything you want.

Assigning Values to Variables

- Now that we've declared our variable names, we need to give them values.
- If you declare a variable without assigning any value to it, its value is `undefined`.
- `var name;`
- `// => undefined`
-
- So, let's try assigning a value to a variable:
- `var name = 'Bill'; // the variable "name" gets assigned the string 'Bill'`
-
- `name;`
- `=> 'Bill' // the variable "name" now is the same as writing the string 'Bill'`
-
- Notice that the value is a word (not a number), so we put single quotation marks around it.
- We always put quotation marks around strings (values that consists of letters and/or other characters). We'll be talking more about strings later in this lesson.
- **Note:** The `//  =>` the example above shows what the output will be when you type the preceding line into the console and hit "enter." For exercises and checkpoints, you will never have to actually type out any lines with `//  =>` in the console.

Variable Syntax

- `var name = "Bill";`
-
- Notice the space before and after the = sign and the semicolon after the string.
- Making sure these are in place is a good habit to develop and will be important when your code gets more complicated later on.

Variable Syntax (Continued)

- JS is a programming language, and like any language, it has its own grammar and rules of operation. Let's look at four rules of thumb:

1. When creating/declaring a variable for the first time, use the `var` keyword.
2. Variable names should be written in camelCase

- The first letter of the first word should be lowercase and the first letter of any subsequent words should be uppercase.

3. Add a space before and after the equals sign = .
4. Statements need to end with a semicolon `;`.
5. Note that in JavaScript, the equals sign doesn't evaluate things the way it does in math. Rather, it *assigns values* to elements, like variables.
6. We call this an *assignment operator*.

Reassigning Variables

- JavaScript runs synchronously and top-down, meaning it updates itself with the latest information given (on the bottom).
- So, the values we give our variables furthest down will overwrite any that have been previously given.
- Let's take a look at a brief example. Say we want to create variables for a character's name, age, and location.
- `var name = "Marty McFly";`
- `var age = 16;`
- `var location = "Hill Valley";`

Reassigning Variables (Continued)

- We can later replace, or reassign, the values of these variables like so:
- `var name = "Marty McFly";`
- `var age = 16;`
- `var location = "Hill Valley";`
-
- `name = "Doc Brown";`
- `age = 65;`
- `location = "Future Town";`
-
-
- On the next slide, you'll find a brief video summarizing how to assign and reassign variables.

Video: Assigning Variables

- MyGA | General Assembly

Syntax Guidelines

- A good habit to help you avoid coding headaches: Mind your syntax.
- Here are some syntax rules that are crucial to remember:
- **JS is case-sensitive**
- For example: `numberofstudents` is not the same as `numberOfStudents`, which is not the same as `NuMbErOfStUdEnTs`.
- When you want to use a variable name consisting of several words, you'll get an error if you have spaces between the words. Instead, you can combine all of the words together into one long variable name. The first letter in this variable name should be lowercase, and the first letter in each word that follows should be uppercase. This is called camelCase. (See how it resembles a camel with humps?)
- Here's another example:
- `var camelsAreAwesome = true;`

Syntax Guidelines (Continued)

- **End statements with a semicolon (;).**
- `x = x + 1;`
- After each line of instruction, be sure to use a semicolon. Although you may be able to get away without using semicolons in some browsers, JavaScript often does crazy things when it guesses where to put semicolons itself. It's better to be explicit and tell the interpreter exactly where a statement should end.

Exercise

- Ready for some hands-on practice? Here's what to do for each step of this exercise:
- Open the JS Bin Console in Chrome.
- Type your JavaScript into the "Console" tab.
- Hit the "return" key to run that line of code.
- You can click the "Clear" button to clear the console when you want a clean slate.
-

Step 1

- Create the following variables (resist the urge to copy/paste and practice typing these out!):
- `var name = 'Susan Smith';`
- `var age = 20;`
- `var hometown = "Hawaii";`

Step 1 (Continued)

- You may see `undefined` displayed in the console after declaring each variable and hitting the "return" key:
- `var name = 'Susan Smith';`
- `// => undefined`
-
- Don't worry too much about this for now, but be aware that this is the expected behavior.
- **Hint:** You may have noticed that each of those lines ended in a `;` — in JavaScript, a semicolon is used to denote the end of a line. Although your code may execute without them, there may be cases in which a missing semicolon could cause unexpected results. It's best to just get in the habit of using them.

Step 2

- Now try checking, or accessing, the values of the three variables by typing each variable name and hitting "enter:"
- `name;`
- `age;`
- `hometown;`

Step 3

- Then, update (reassign) the value of each variable to your own first and last name, your own age, and your own location.
- For example:
- `name = 'Amy Hill';`
- `age = 25;`
- `hometown = "San Francisco";`

Step 4

- Again, try checking, or accessing, the values of the three variables by typing each variable name and hitting "enter:"
- `name;`
- `age;`
- `hometown;`

Summary

- We can redefine our variables `name`, `age`, and `hometown` as many times as we want. However, ***only the most recent value of each variable is retained*** Once a variable is redefined, its original value is lost forever.
- Consider the following JavaScript code:
- `var x = 1;`
- `x;`
- `// => 1`
- `x = 2 + x;`
- `// => 3`
- `x;`
- `// => 3`
-
- On the first line, we are *assigning* the variable `x`, setting it equal to the integer `1`.
- Then, on the next line, we are *reassigning* the variable `x`, setting it equal to: `2`plus the most recent value of x (in this case, `1`).
- `x` would now be equal to the integer `3`.
-

Summary (Continued)

- Suppose we ran the following lines of code in order, one by one.
- `var x = 10;`
- `x = 1;`
- `x = 5;`
- `x = 15;`
- `x + 2;`
-
- What does that last line evaluate to? Or, put differently, what is the most recent value of `x` (as of line 4) + `2`?
- If you guessed `17`, you're correct!
- Is `x`'s value now `17`?
- Not so fast! The last time a value was assigned to `x` was on line 4, so `x` is still `15`.

Conclusion

- Phew, we covered a lot in this lesson! We defined variables, learned variable syntax, discussed how to assign and reassign variables, and reviewed syntax guidelines.
- Feel free to go back and review this material as necessary.