Variables and Operators

Variables store data

```
lastName = 'Jones';
var firstName;
const age = 50;
let hasADog = true;
```

scoping

var, let, and const

Scoping refers to where a variable can be seen

Global scope

Function scope

Block scope

```
x = "foo";
y = 5;
```

Variables are global by default

To give them function scope ...

1. Put them anywhere in a function
2. Use var

To give them block scope ...

1. Put them in a block
2. Use let or const

```
let foo;
let bar = 5;
const baz = 10;
```

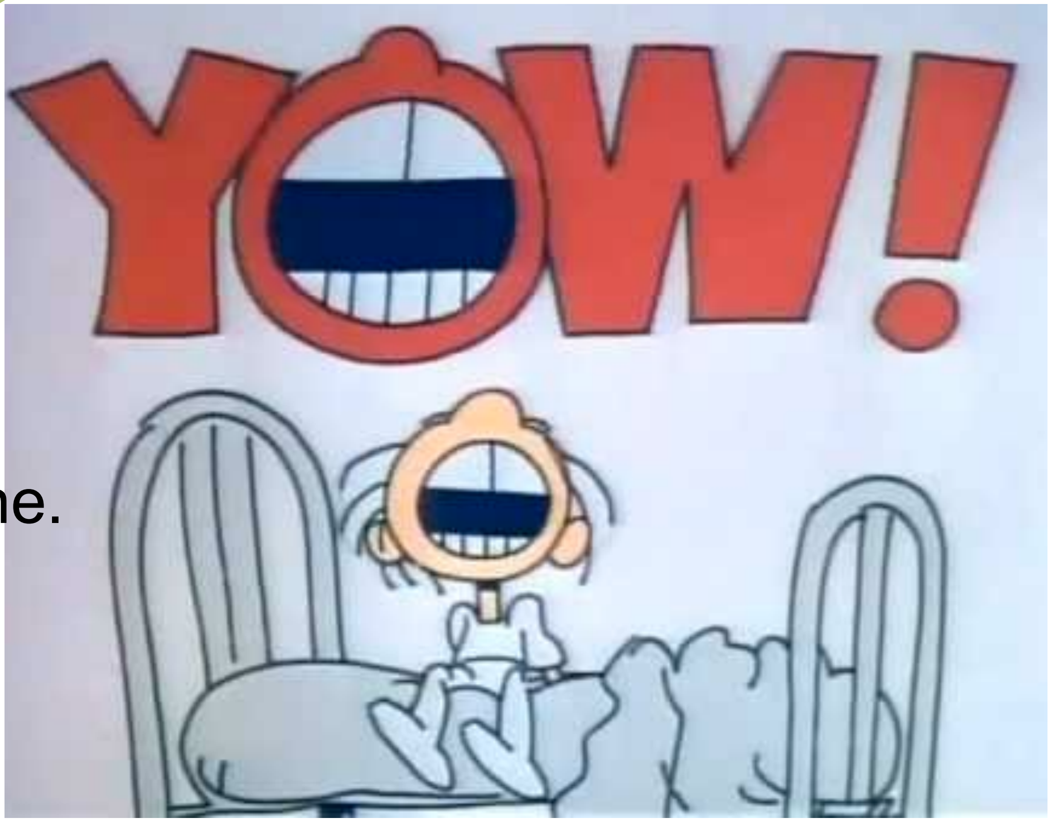
ES 2015

For example ...

```
function foo() {
  x = 5;
  if (someCondition) {
    var x;
  }
  x = 10;
}
```

All of these variables are the same instance!

Unexpected and error-prone.



New way

- let is like var, but it honors block scope and is not hoisted.

```
if (somethingTruthy) {
  let foo='10';  // foo is block-scoped
}
// foo is not defined here
```

let isn't hoisted

- This doesn't work

```
console.log(bar);
// referenceError; bar isn't defined
let bar="value";
```

- This is okay, though

```
function readThere () {
  return there
}
let there = 'foo'
console.log(readThere())
```

const behaves just like let

- block-scoped
- Not hoisted
- Except ...

const is constant ...

- Values must be assigned on declaration:

```
const x = 10;

x='foo'; // throws TypeError

const x = 'foo'; // throws -- redeclaration
```

... except when it's not

```
const simpsons = ['homer', 'marge', 'bart', 'lisa'];
simpsons.push('maggie');  // totally works.
const neighbor = {
  first: "Ned"
}
neighbor.kids = ["Rod", "Todd"];  // Also works.
```

const is not immutable

To use a non-existent variable is a fatal error

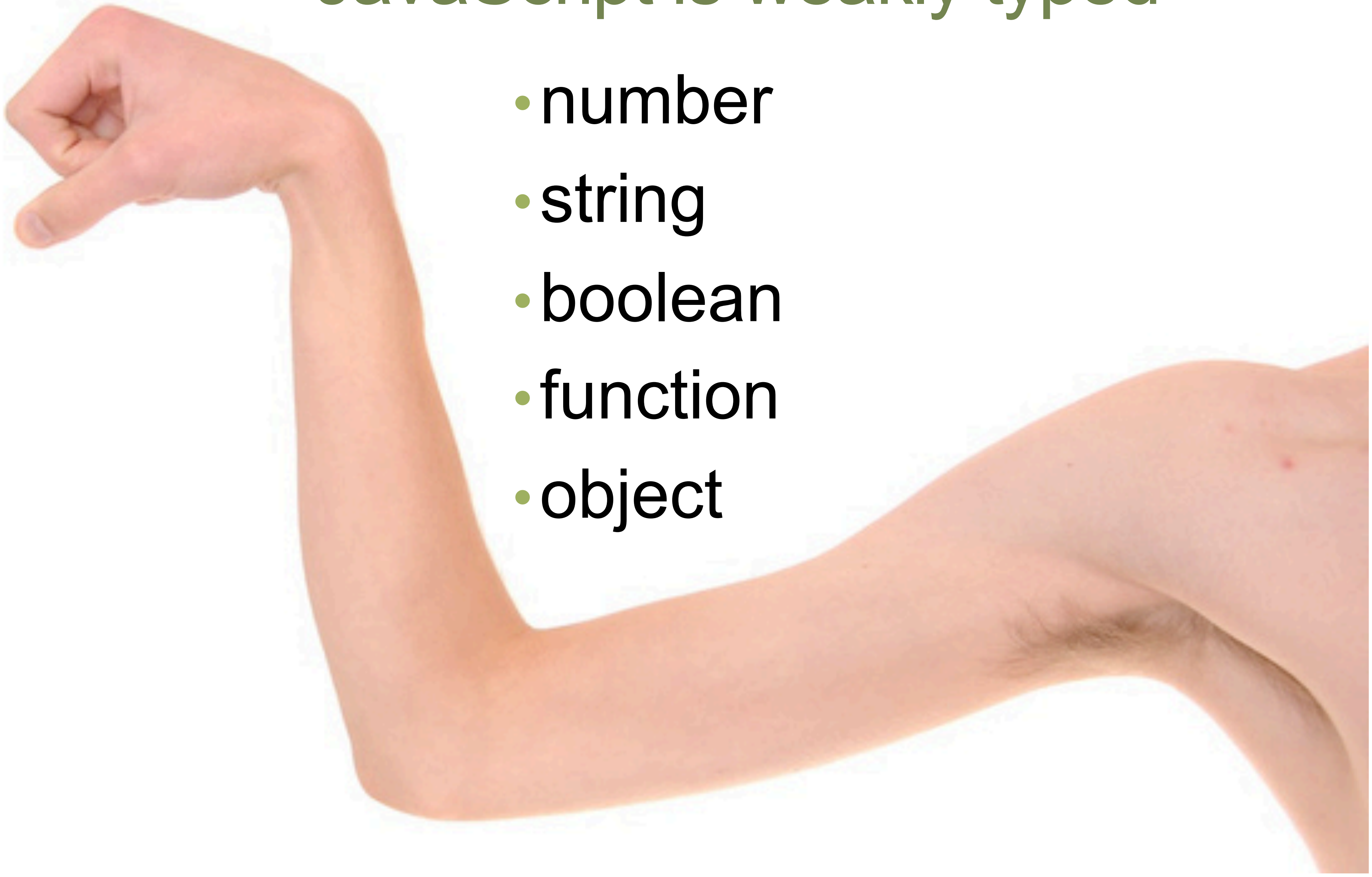
- To use a non-existent variable is a fatal error.

```
let x = 5;

let z = x + y;  //ReferenceError: y is not defined
```

Data types

JavaScript is weakly typed



- number
- string
- boolean
- function
- object

How do I know what I'm working with?

- The typeof operator

```
function foo(x) {
  if (typeof x != "object")
    throw new Exception("I need an object");
  else
    // Do stuff with that object
}
```

Numbers

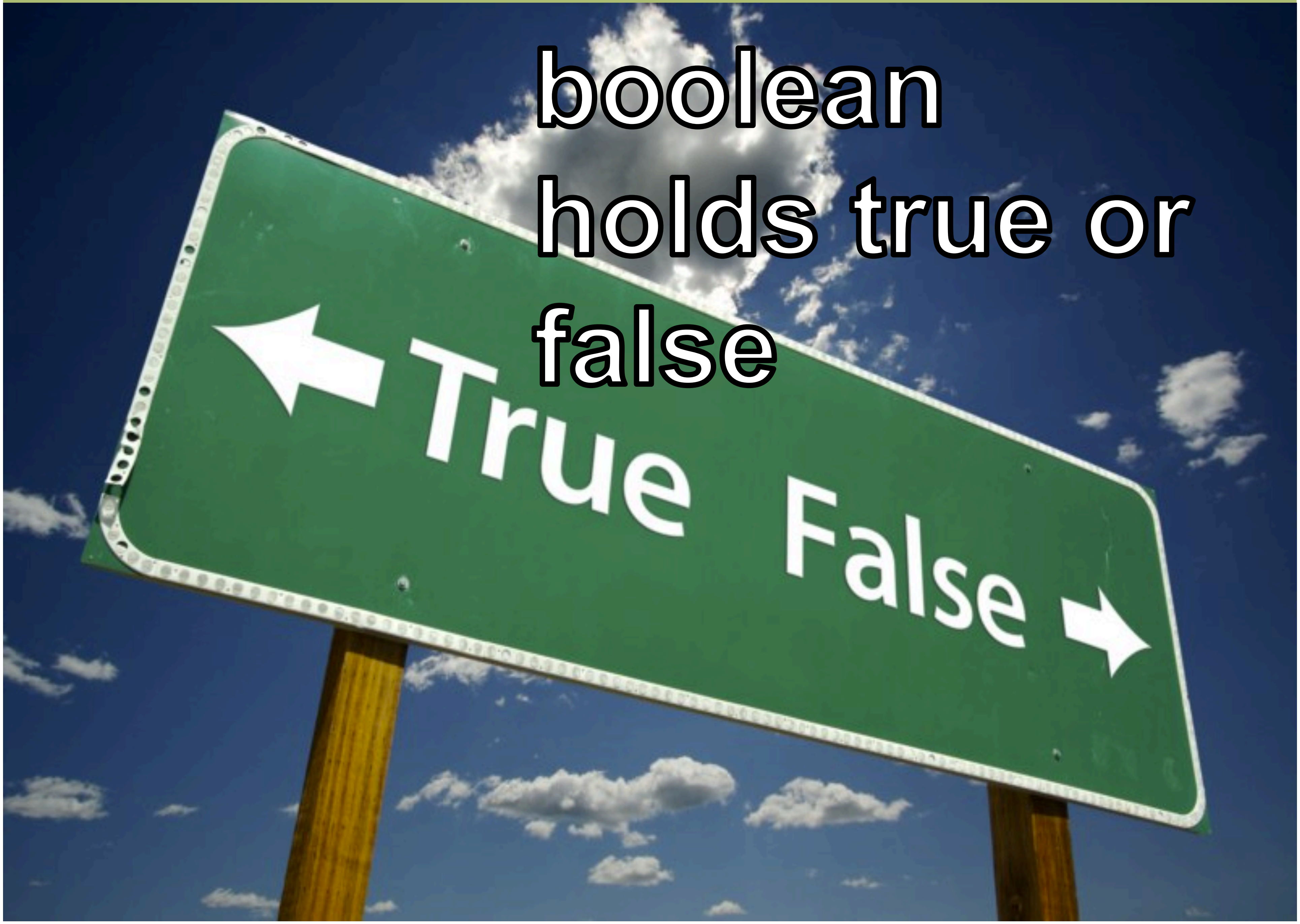
- Numbers are IEEE754 double-precision floating point numbers – Max size is 1.8 E 308 or 9 quintillion

```
alert([Number.MAX_VALUE, Number.MIN_VALUE]);
```

```
var x = 5;         // Integer
var x = 5.5;       // Floating point
var x = 0o50;      // Octal
var x = 0x37;      // Hexadecimal
var x = 0b110111;  // Binary
var x = 5.0e50;    // Scientific notation
```

Numbers are very flexible

boolean
holds true or
false

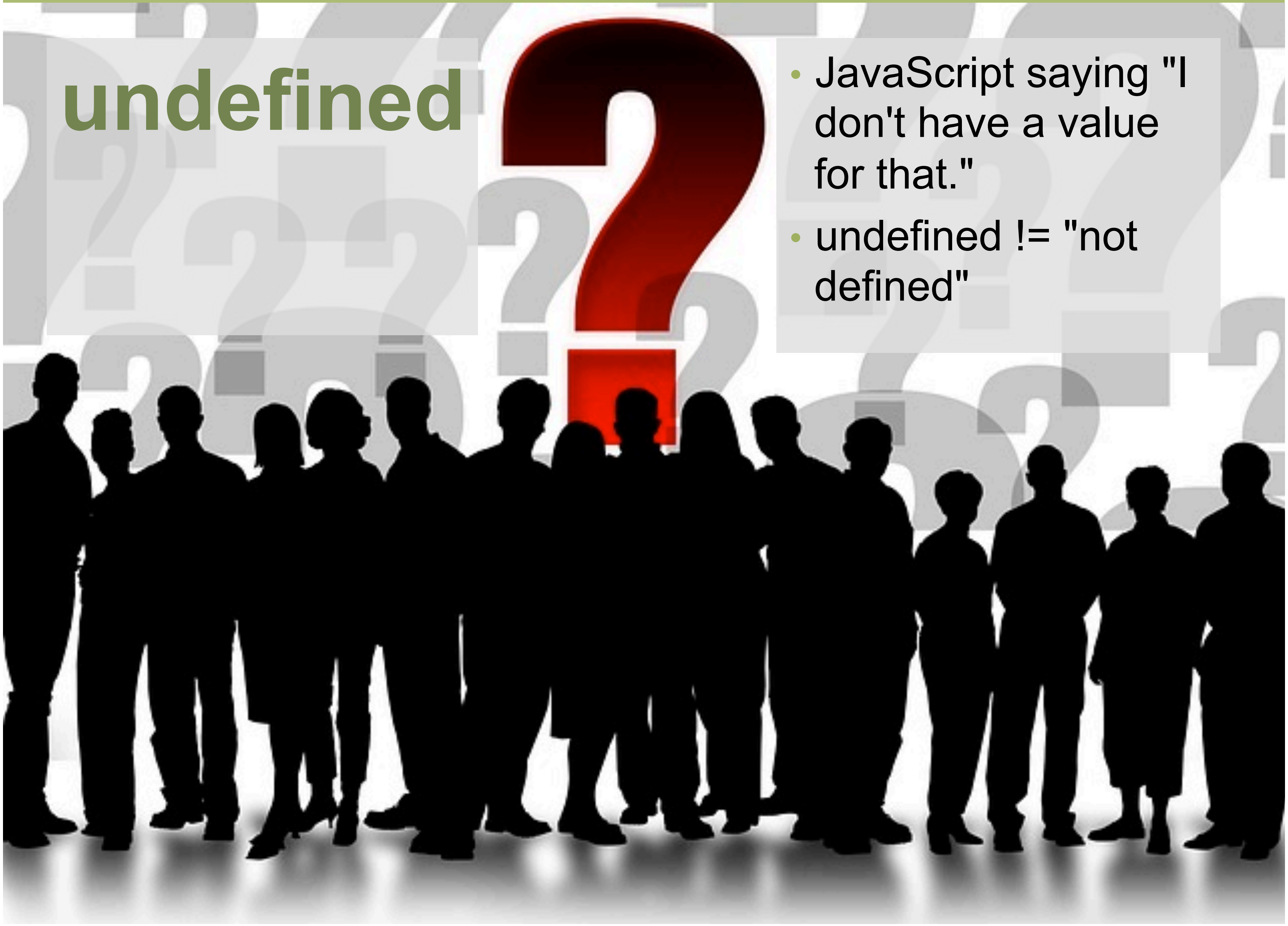


JavaScript Objects are simple hashes

- Merely a collection/hash/dictionary of key-value pairs. Very simple.

firstName	"Rufus"
middleName	"Xavier"
lastName	"Sarsparilla"
age	10
sister	rafaela
pet	rhinoceros

- Not based on classes at all.



undefined

- JavaScript saying "I don't have a value for that."
- undefined != "not defined"

"Truthiness" = coercing to boolean

```
var x = _______;
if (x) console.log('true'); else console.log('false');
```

If x is ...	console says ...
`true`	true
`false`	false
`"any string"`	true
`""`	false
`"false"`	true
`100`	true
`0`	false
`100/0`	true
`null`	false
`undefined`	false
`{}`	true

Arrays

Creating Arrays

```
days = new Array();  // Not recommended, but works
days = [];
days = ['Mon', 'Tues', 'Wed', 'Thu', 'Fri' ];
```

- Note that you don't specify a size

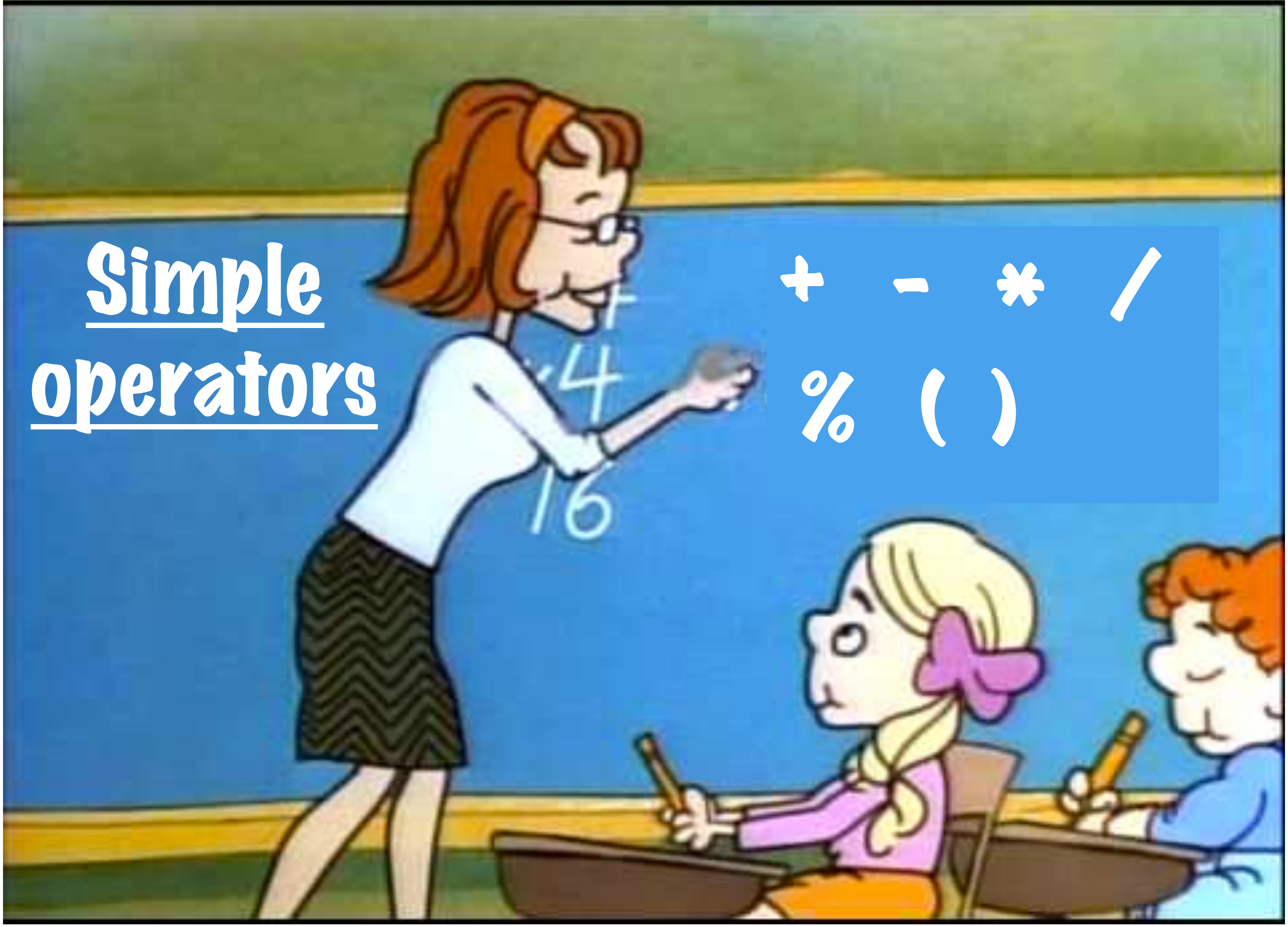
Reading and writing arrays

```
x = days[0];  //Mon
y = 1;
x = days[y+2];  //Thu
x = days[days.length-1]; //The last element
// Arrays are sparse, not dense
days[54] = "";  // Now days.length=55
x = days[1000];  // Not an error!! Merely undefined
```

Adding to & removing from arrays

```
let numElements = a.push(newVal);
let thingRemoved = a.pop();
let numElements = a.unshift(newVal);
let thingRemoved = a.shift();
```

Operators



Simple operators

+ - * /

% ( )

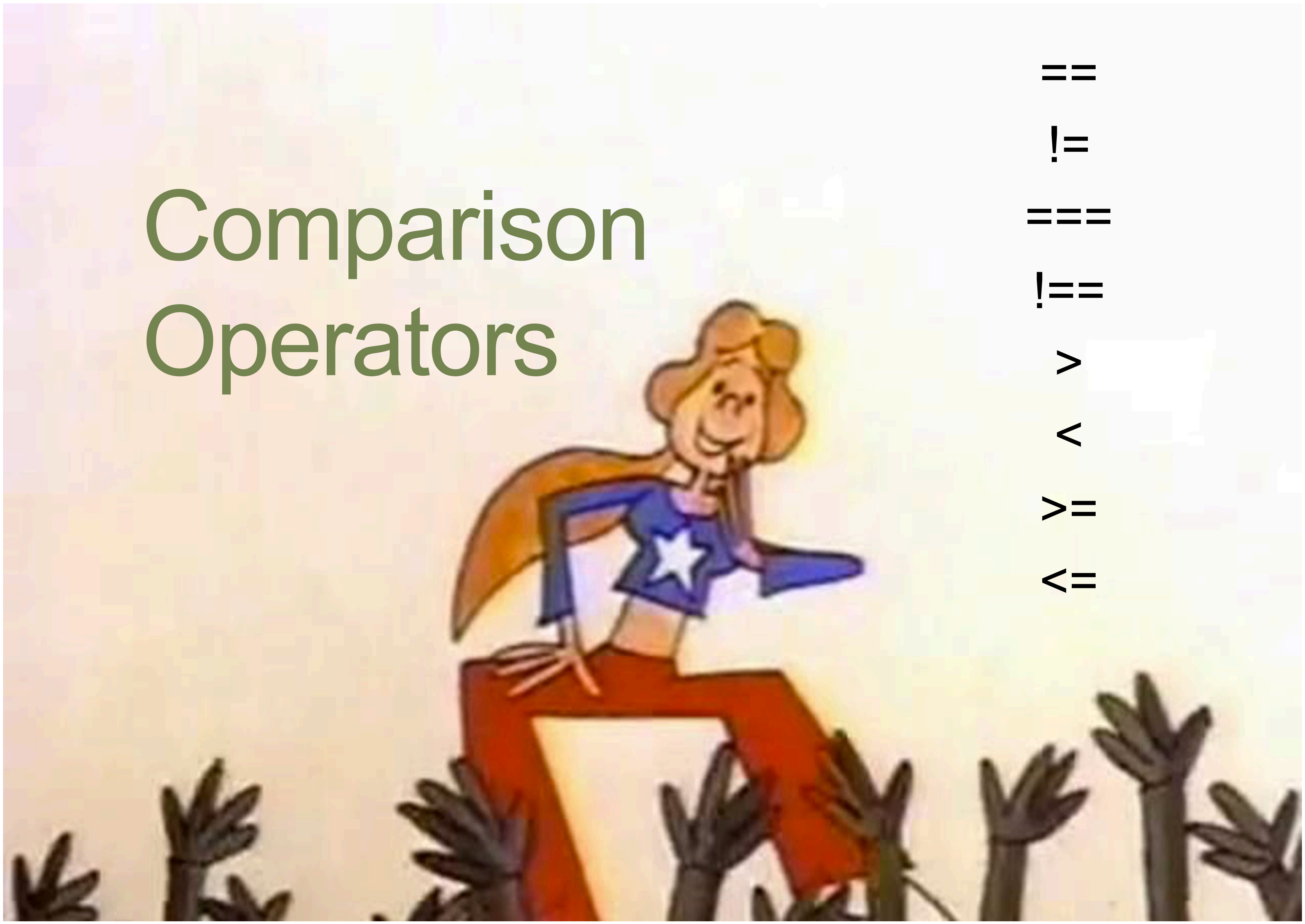
Combining numbers and strings

```
var n = "3";
var x = "The magic number is " + n;
// implicitly coerces numbers to strings
x = 5 + n;    // 53
x = 5 + Number(n);  // 8
x = 5 + +n;    // 8
```

Auto-assignment operators let you write quicker

```
x += 5; same as x = x + 5;
x -= 5; same as x = x - 5;
x *= 5; same as x = x * 5;
x /= 5; same as x = x / 5;
x++; same as x = x + 1;
x--; same as x = x - 1;
```

Comparison Operators



==

!=

===

!==

>

<

>=

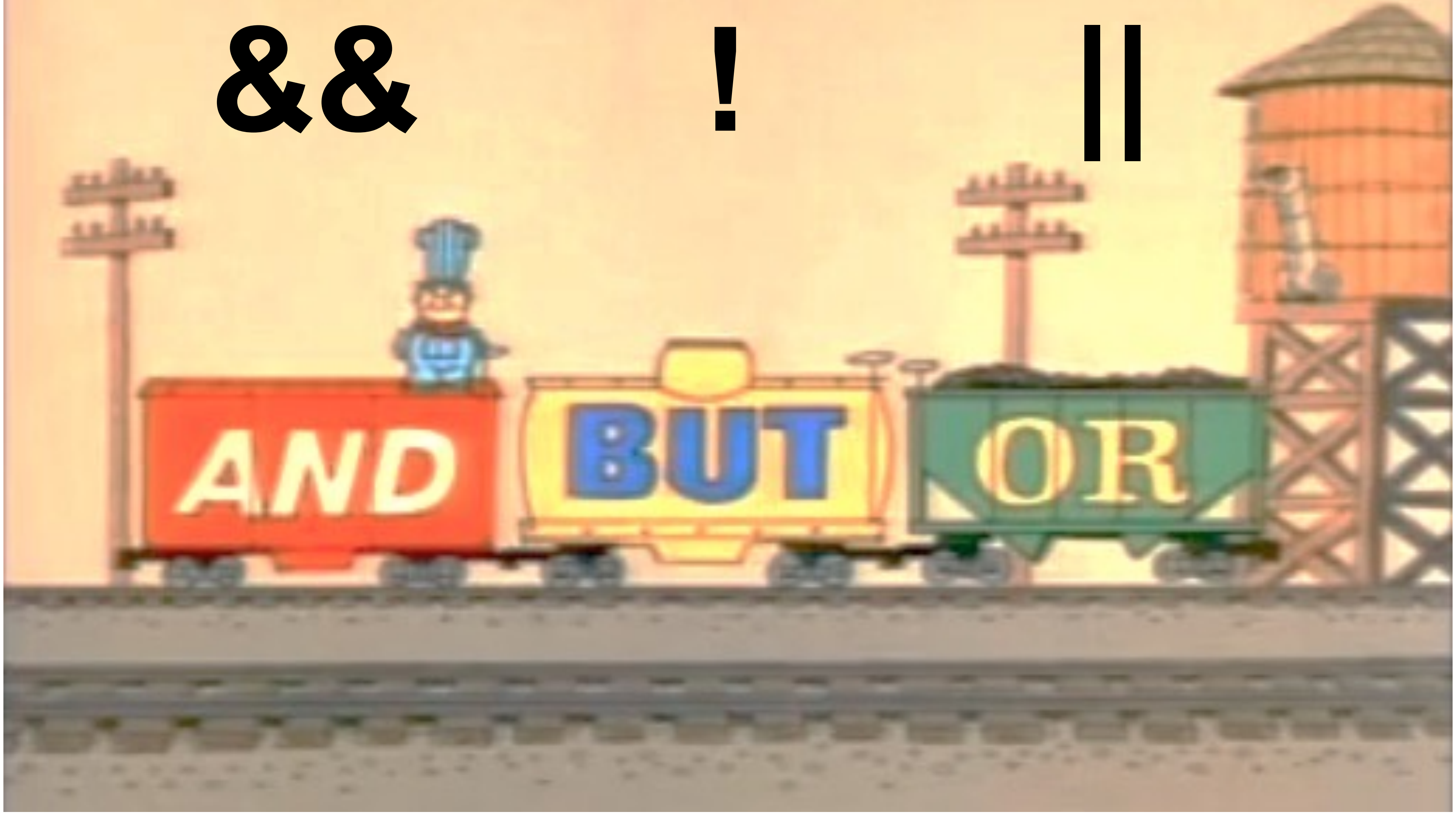
<=

Identical objects are never equal

```
var o1 = { foo: "bar" };
var o2 = { foo: "bar" };
o1 == o2  // false
o1 === o2  // false
var o3 = o1;
o1 == o3  // true
o1 === o3 // true
```

Boolean operators

&& ! ||



tl;dr

- JavaScript has ...
- Dynamically-typed values
  - Numbers
  - Strings
  - Booleans
  - Functions
  - Objects
- Arrays
- Mathematical operators like +, -, *, /, ()
- Auto operators like +=, -=, *=, ++, --
- Comparison operators like ==, ===, !=, >, <, >=
- Logical operators like &&, !, and ||