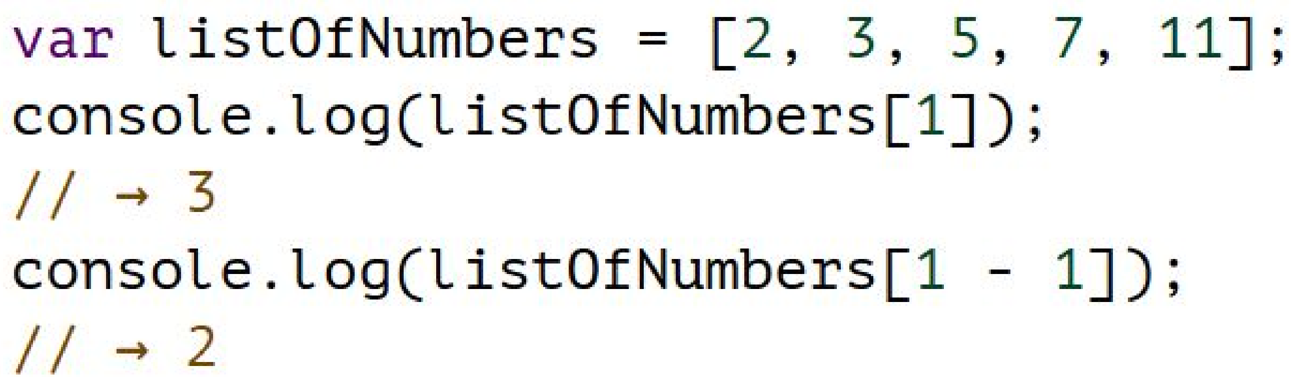
JavaScript IV

Data Structures: Objects and Arrays

Arrays

- Written as a list of values between square brackets, separated by commas
- Use square brackets to get element at an index
- Arrays are objects



```
var listOfNumbers = [2, 3, 5, 7, 11];
console.log(listOfNumbers[1]);
// → 3
console.log(listOfNumbers[1 - 1]);
// → 2
```

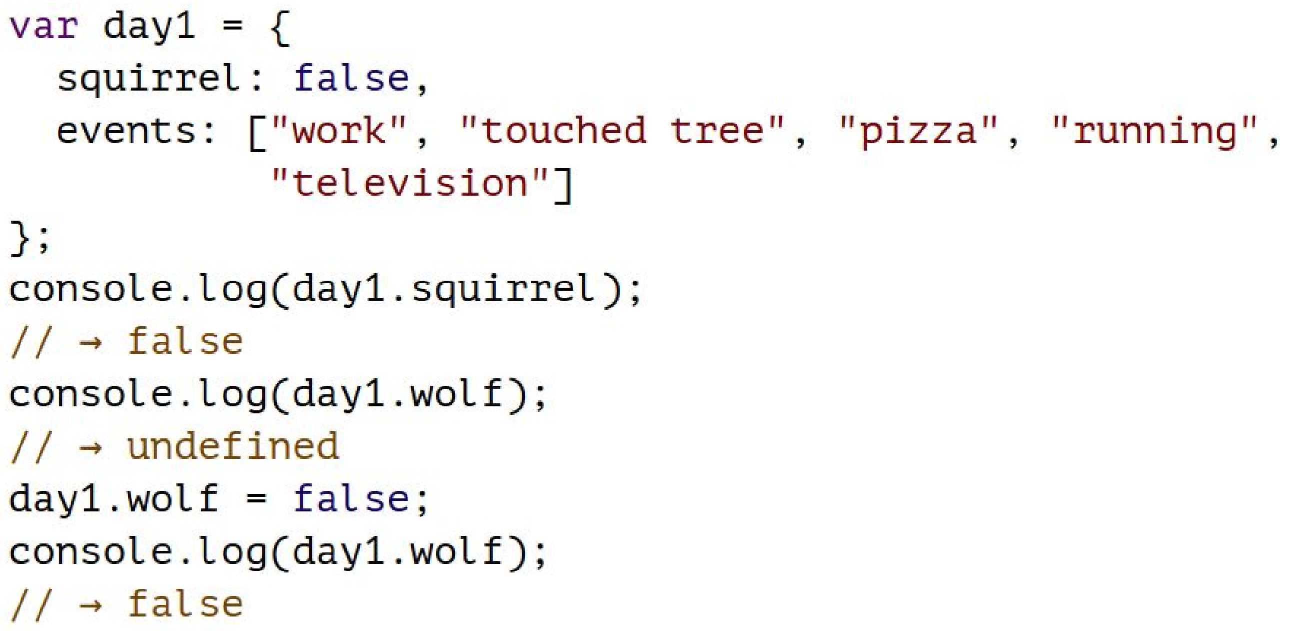
Objects: Arbitrary collections of properties

Property names: stuff before the colon

Property values: stuff after the semicolon

Properties whose names are not valid variable names have to be quoted

Curly braces have two meanings: a block of code or an object



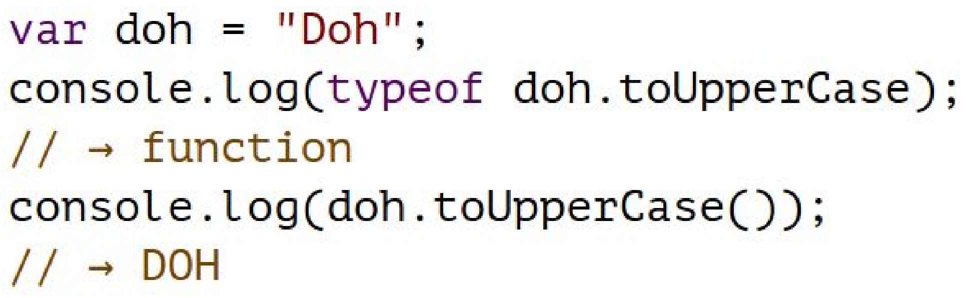
```
var day1 = {
  squirrel: false,
  events: ["work", "touched tree", "pizza", "running",
           "television"]
};
console.log(day1.squirrel);
// → false
console.log(day1.wolf);
// → undefined
day1.wolf = false;
console.log(day1.wolf);
// → false
```

Properties

- Almost all JavaScript values have properties
- myString.length, Math.max
- Access properties using the dot (value.x) or brackets (value[x])
    - Using dot: must be valid variable name
    - Using brackets: expression in brackets evaluated to get property name (even if it isn't a valid variable name)
    - E.g. `{"Froot Loops Toucan": 12, time: 23455032459}.` To access Froot Loops Toucan property, you need the brackets .

Methods: Functions as Properties

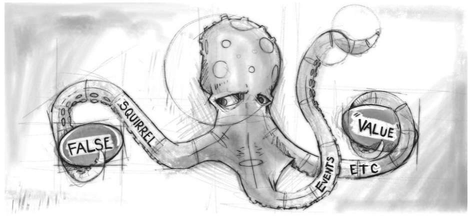
- Console.log, a string’s toUpperCase properties

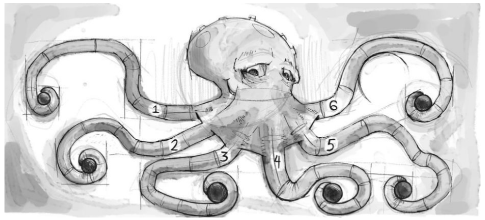


```
var doh = "Doh";
console.log(typeof doh.toUpperCase);
// → function
console.log(doh.toUpperCase());
// → DOH
```

- More examples: the `push` and `pop` methods of arrays

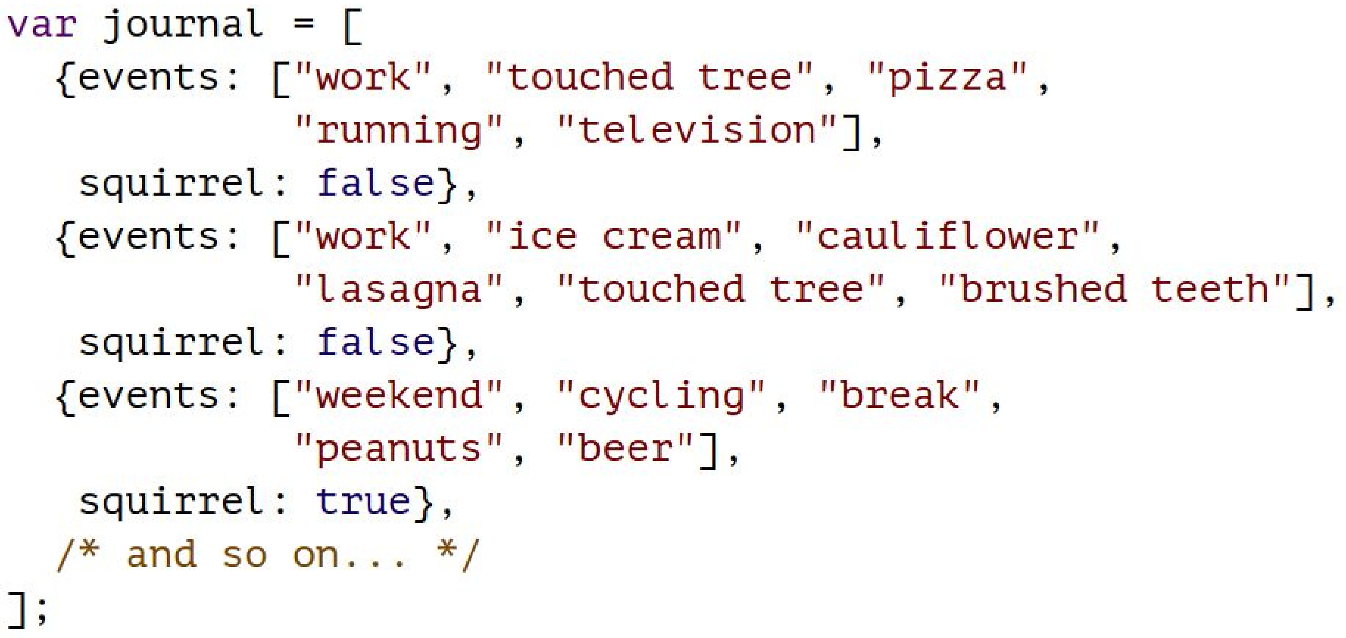
Property Bindings





- Similar to variables: they grasp values
- Multiple properties might bind the same value
- `delete` removes a property binding
- Arrays are specialized objects that store sequences of things (see octopus 2)

An Array of Objects

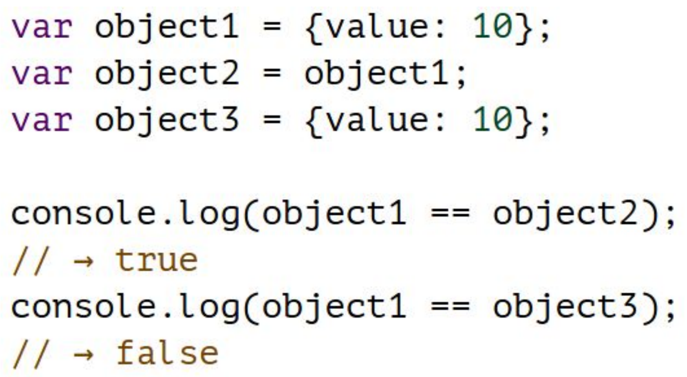


```
var journal = [
  {events: ["work", "touched tree", "pizza",
            "running", "television"],
   squirrel: false},
  {events: ["work", "ice cream", "cauliflower",
            "lasagna", "touched tree", "brushed teeth"],
   squirrel: false},
  {events: ["weekend", "cycling", "break",
            "peanuts", "beer"],
   squirrel: true},
  /* and so on... */
];
```

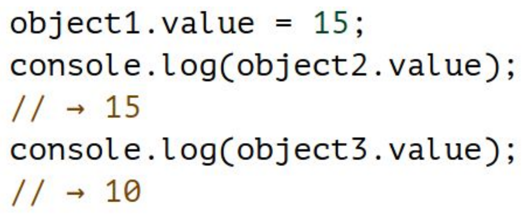
Mutability

There is a difference between two variables referring to the same object and two variables referring to two objects with the same contents

- Earlier values are all immutable: they cannot change
- Objects, however, are mutable --- one can change properties of an object
- There are no *deep* comparisons built into JavaScript



```
var object1 = {value: 10};
var object2 = object1;
var object3 = {value: 10};

console.log(object1 == object2);
// → true
console.log(object1 == object3);
// → false
```



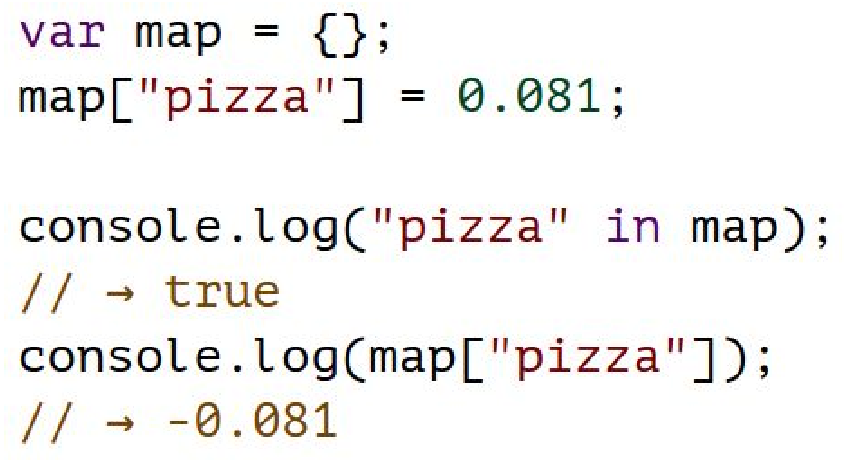
```
object1.value = 15;
console.log(object2.value);
// → 15
console.log(object3.value);
// → 10
```

Objects as Maps

- A map “maps” keys to values; it’s a group of key/value pairs

- Faster than searching for a value in an array

- Sometimes called *dictionaries*



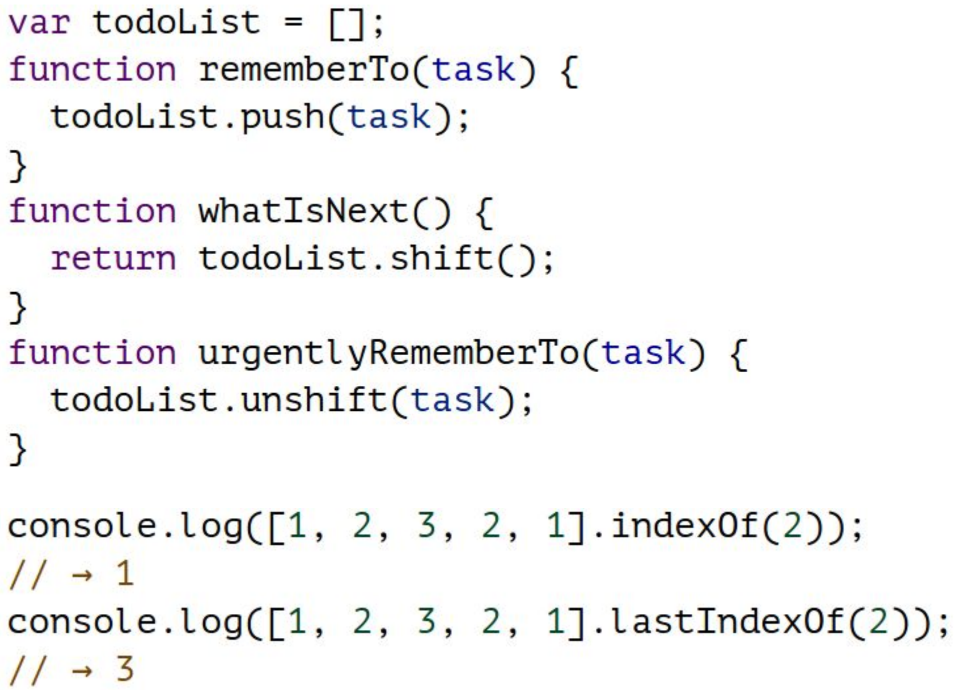
```
var map = {};
map["pizza"] = 0.081;

console.log("pizza" in map);
// → true
console.log(map["pizza"]);
// → -0.081
```

Array Methods

`push, pop`: adds or subtracts an element from the end of an array

`unshift, shift`: adds/removes elements from the start of an array

`indexOf, lastIndexOf` searches for a value and returns the index where it was found

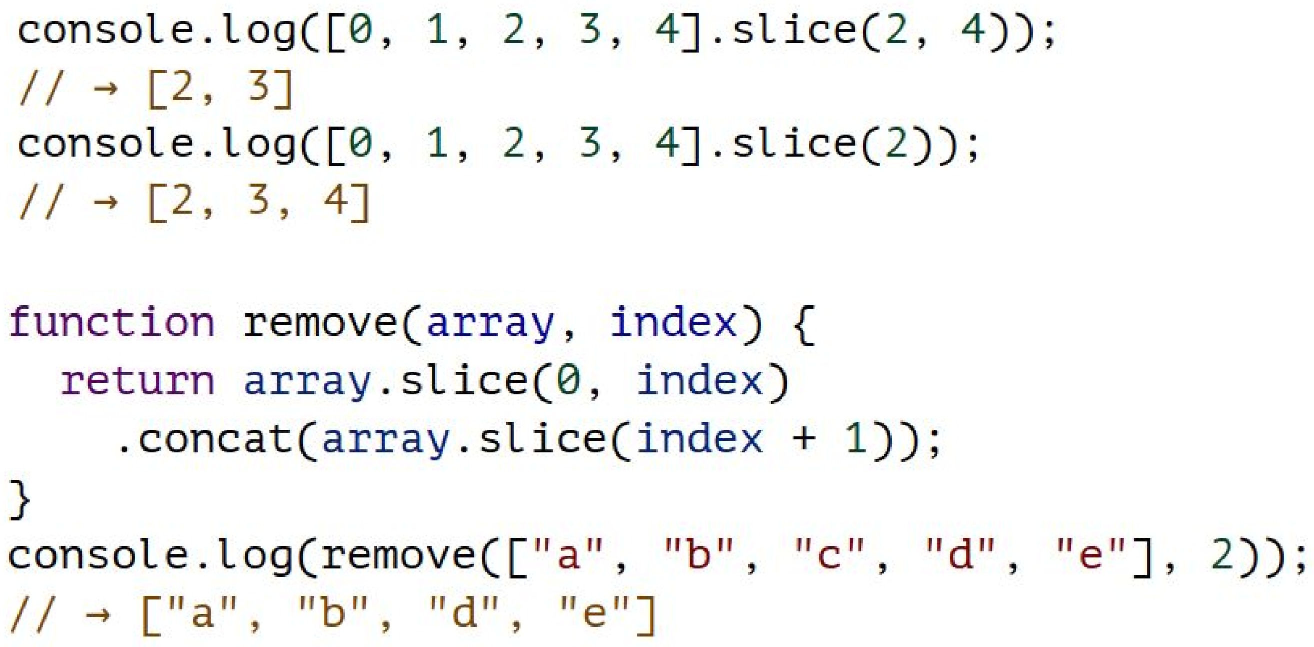


```
var todoList = [];
function rememberTo(task) {
  todoList.push(task);
}
function whatIsNext() {
  return todoList.shift();
}
function urgentlyRememberTo(task) {
  todoList.unshift(task);
}

console.log([1, 2, 3, 2, 1].indexOf(2));
// → 1
console.log([1, 2, 3, 2, 1].lastIndexOf(2));
// → 3
```

More Array Methods

- Use `slice` to get a subarray

- Use `concat` to combine two arrays



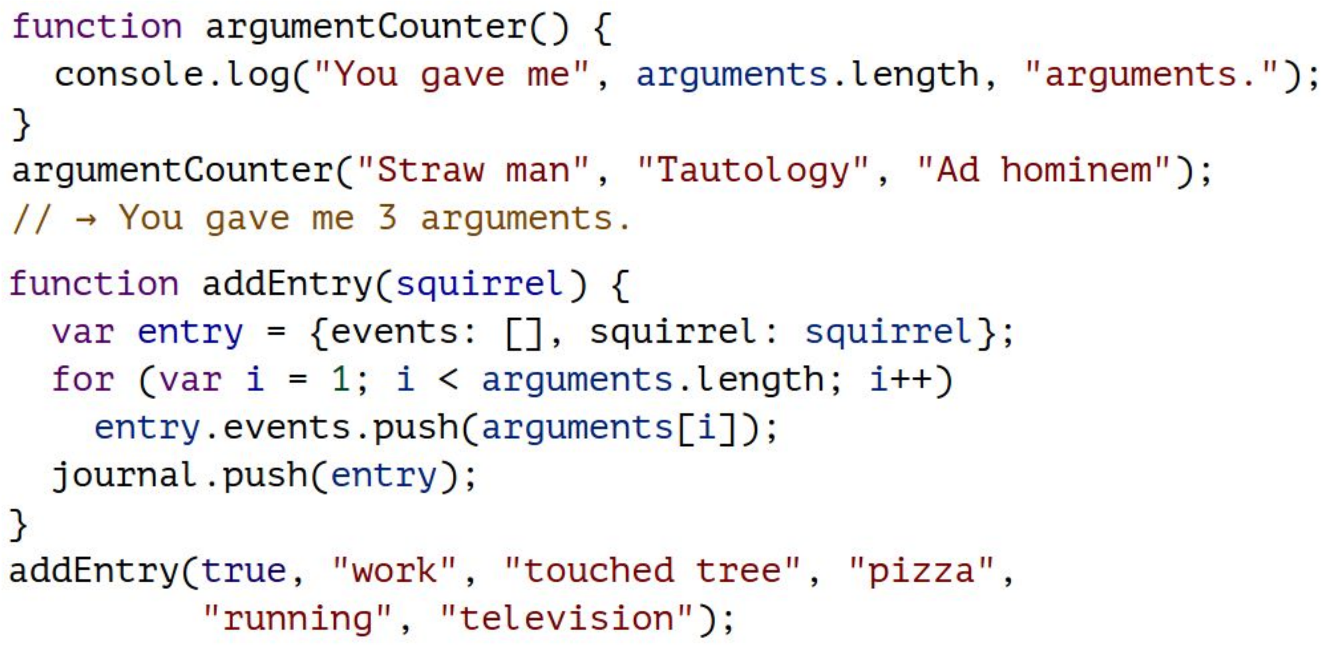
```
console.log([0, 1, 2, 3, 4].slice(2, 4));
// → [2, 3]
console.log([0, 1, 2, 3, 4].slice(2));
// → [2, 3, 4]

function remove(array, index) {
  return array.slice(0, index)
    .concat(array.slice(index + 1));
}
console.log(remove(["a", "b", "c", "d", "e"], 2));
// → ["a", "b", "d", "e"]
```

More on Strings

- Strings are not objects (neither are numbers or booleans)
- Strings are immutable: new properties will not be remembered
- Strings have built-in properties
    - `slice, indexOf, trim, charAt`

The Arguments Object

- Whenever a function is called, a special variable named `arguments` is added to whatever environment the function is in
- It holds all the arguments passed into the function



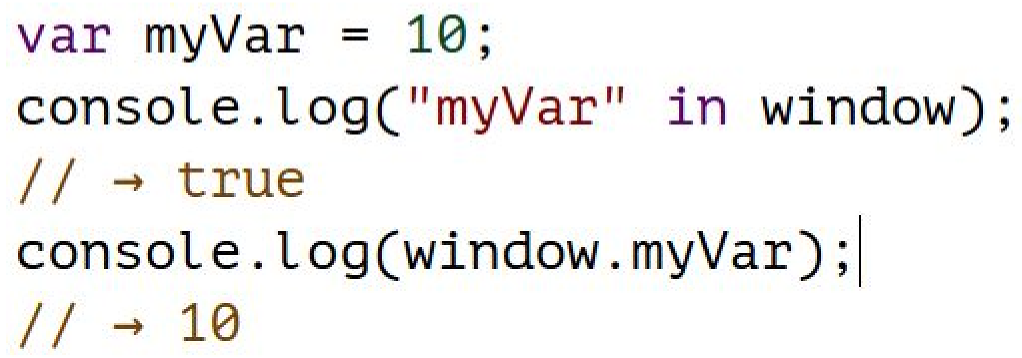
```
function argumentCounter() {
  console.log("You gave me", arguments.length, "arguments.");
}
argumentCounter("Straw man", "Tautology", "Ad hominem");
// → You gave me 3 arguments.

function addEntry(squirrel) {
  var entry = {events: [], squirrel: squirrel};
  for (var i = 1; i < arguments.length; i++)
    entry.events.push(arguments[i]);
  journal.push(entry);
}
addEntry(true, "work", "touched tree", "pizza",
         "running", "television");
```

The Math Object

- A container to hold math-related functions
- It provides a namespace for the functions, so they don’t have to be global variables
    - Javascript does not warn you when you define a variable whose name is taken
- Some functions:
    - Trig and inverse trig functions, PI, max, min, sqrt, pow, random, floor, round, ceil, and others

The Global Object

- The global scope is an object
- Each global variable is a property of this object
- In browsers, `window` is the global object



```
var myVar = 10;
console.log("myVar" in window);
// → true
console.log(window.myVar);
// → 10
```