Core Javascript

Software Systems Development

IIIT Hyderabad

Content

Importance

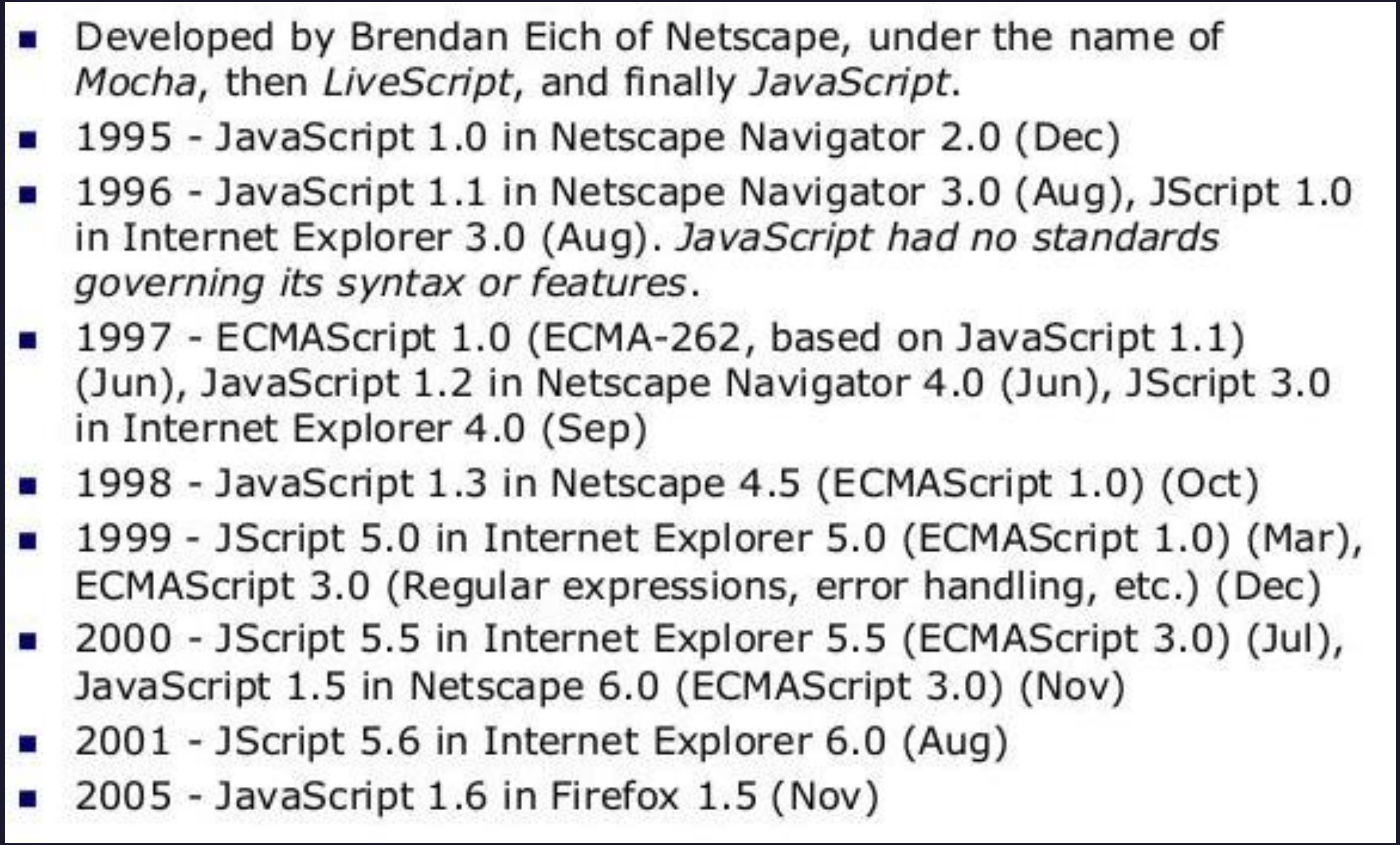
JavaScript is a *cross-platform*, *object-oriented* scripting language used to make webpages interactive (e.g., having complex animations, clickable buttons, popup menus, etc.).

It is a *high-level*, *dynamic*, *untyped* *interpreted* programming language that is well-suited to object-oriented and functional programming styles.

JavaScript is part of the triad of technologies that all Web developers must learn: HTML to specify the content of web pages, CSS to specify the presentation of web pages, and JavaScript to specify the behavior of web pages.

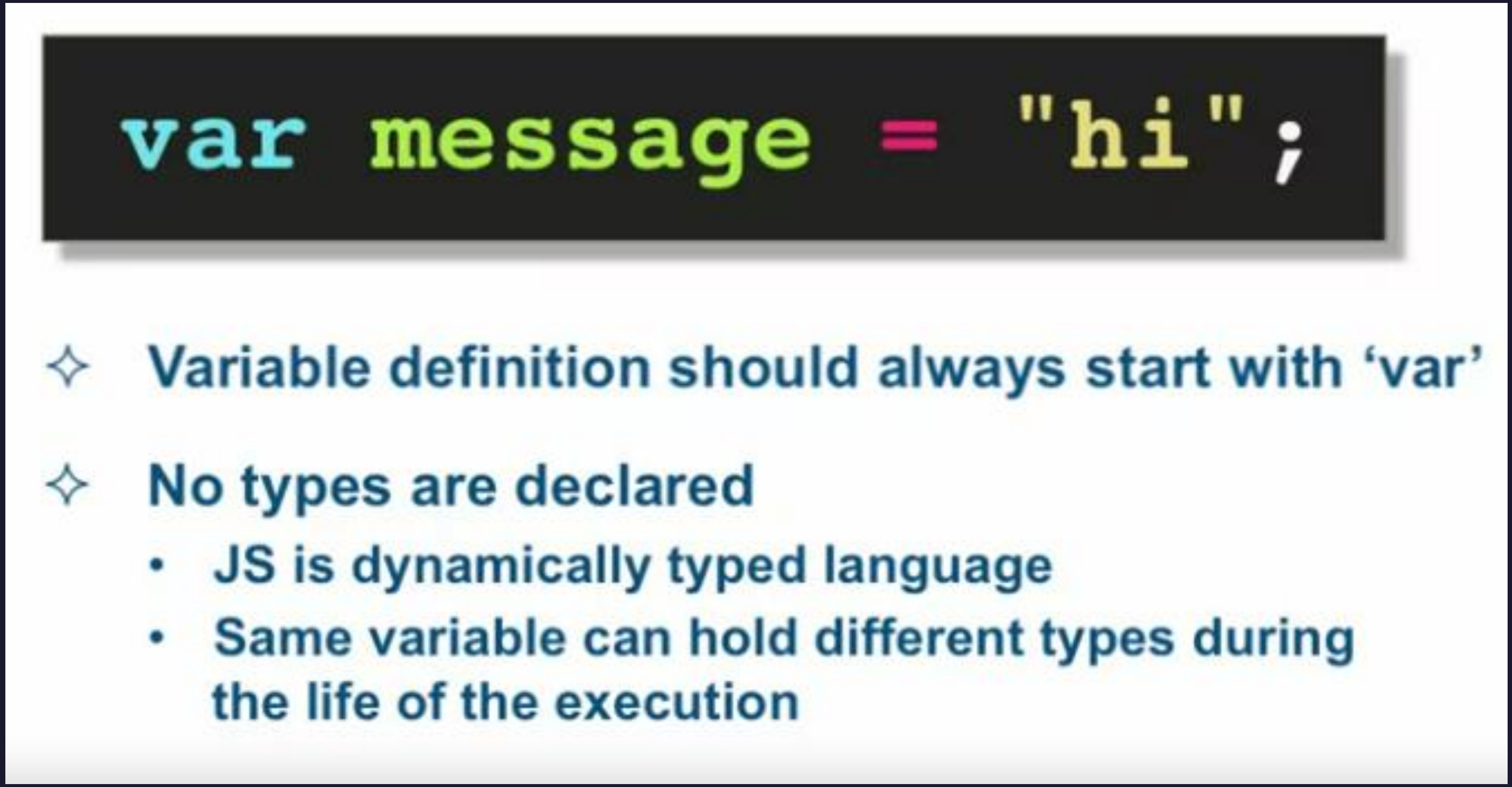
Advanced server side versions of JavaScript such as Node.js, which allow you to add more functionality to a website than downloading files

History



- Developed by Brendan Eich of Netscape, under the name of *Mocha*, then *LiveScript*, and finally *JavaScript*.
- 1995 - JavaScript 1.0 in Netscape Navigator 2.0 (Dec)
- 1996 - JavaScript 1.1 in Netscape Navigator 3.0 (Aug), JScript 1.0 in Internet Explorer 3.0 (Aug). *JavaScript had no standards governing its syntax or features*.
- 1997 - ECMAScript 1.0 (ECMA-262, based on JavaScript 1.1) (Jun), JavaScript 1.2 in Netscape Navigator 4.0 (Jun), JScript 3.0 in Internet Explorer 4.0 (Sep)
- 1998 - JavaScript 1.3 in Netscape 4.5 (ECMAScript 1.0) (Oct)
- 1999 - JScript 5.0 in Internet Explorer 5.0 (ECMAScript 1.0) (Mar), ECMAScript 3.0 (Regular expressions, error handling, etc.) (Dec)
- 2000 - JScript 5.5 in Internet Explorer 5.5 (ECMAScript 3.0) (Jul), JavaScript 1.5 in Netscape 6.0 (ECMAScript 3.0) (Nov)
- 2001 - JScript 5.6 in Internet Explorer 6.0 (Aug)
- 2005 - JavaScript 1.6 in Firefox 1.5 (Nov)

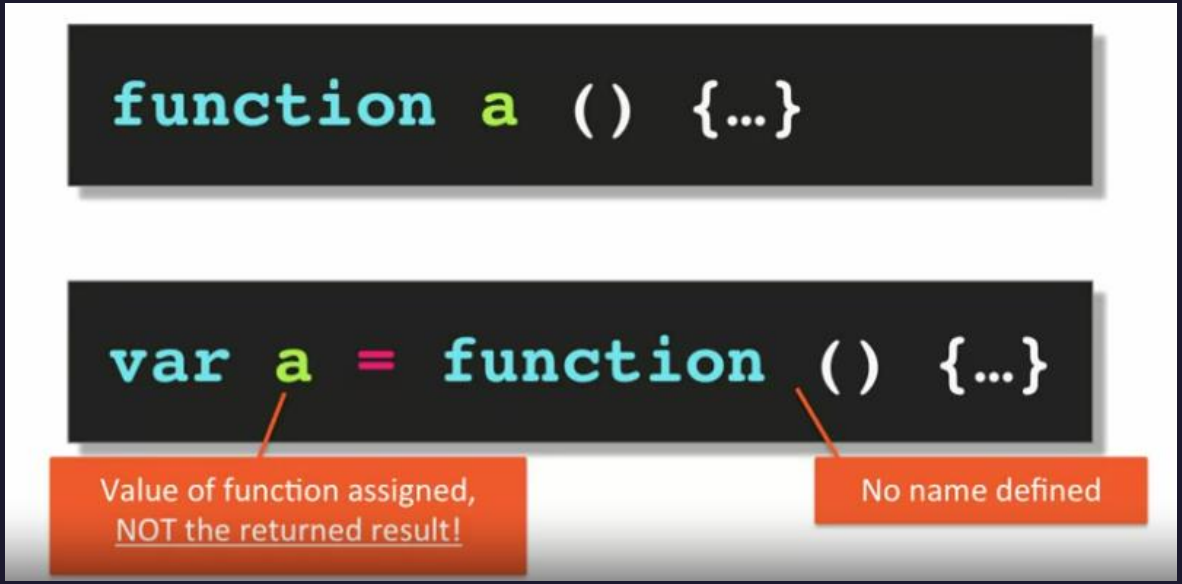
Variables



```
var message = "hi";
```

- ✧ Variable definition should always start with 'var'
- ✧ No types are declared
  - JS is dynamically typed language
  - Same variable can hold different types during the life of the execution

Functions

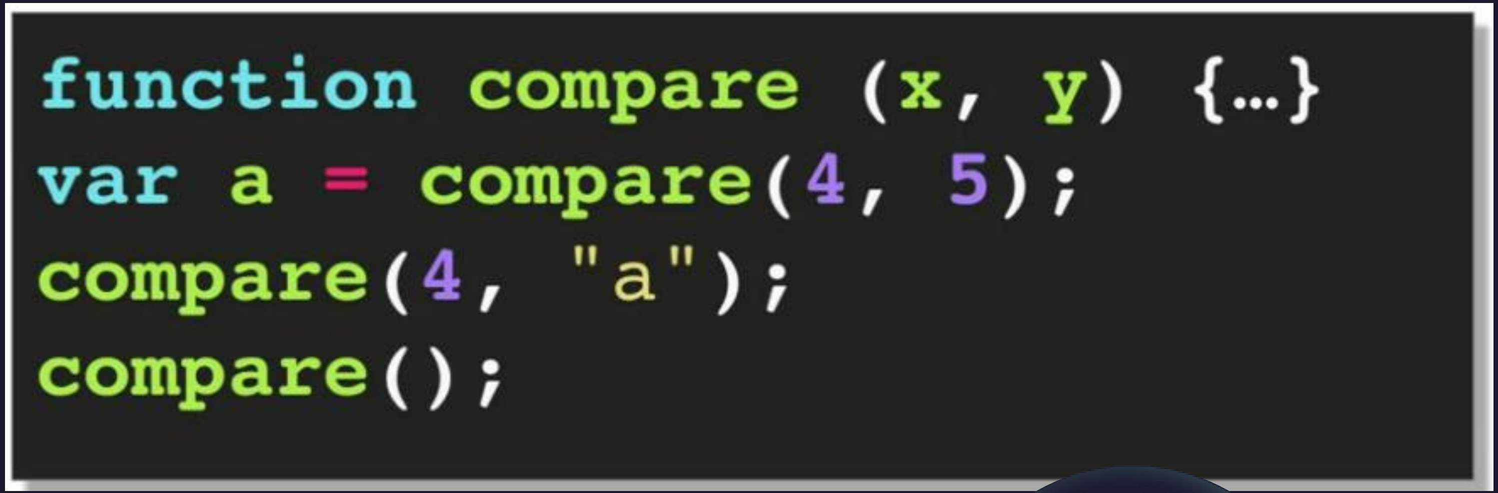


```
function a () {…}
```

```
var a = function () {…}
```

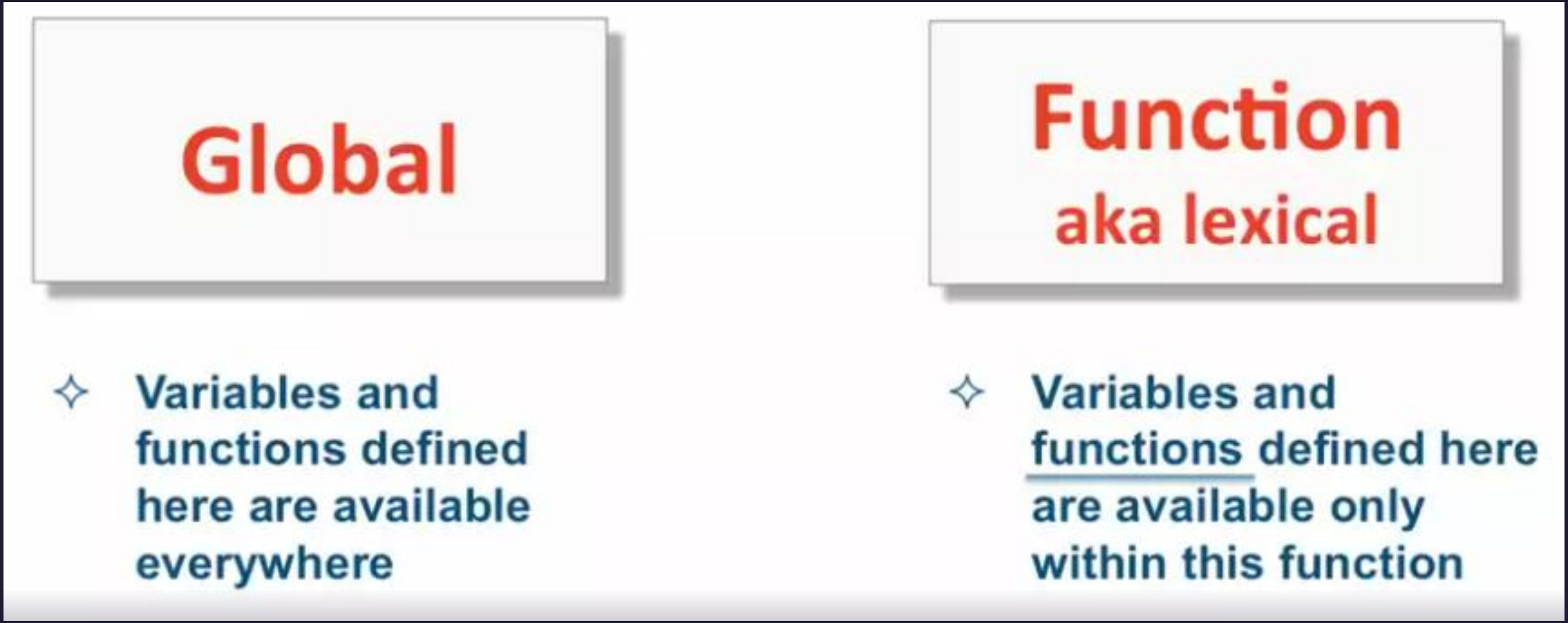
Value of function assigned, NOT the returned result!

No name defined



```
function compare (x, y) {…}
var a = compare(4, 5);
compare(4, "a");
compare();
```

Scope



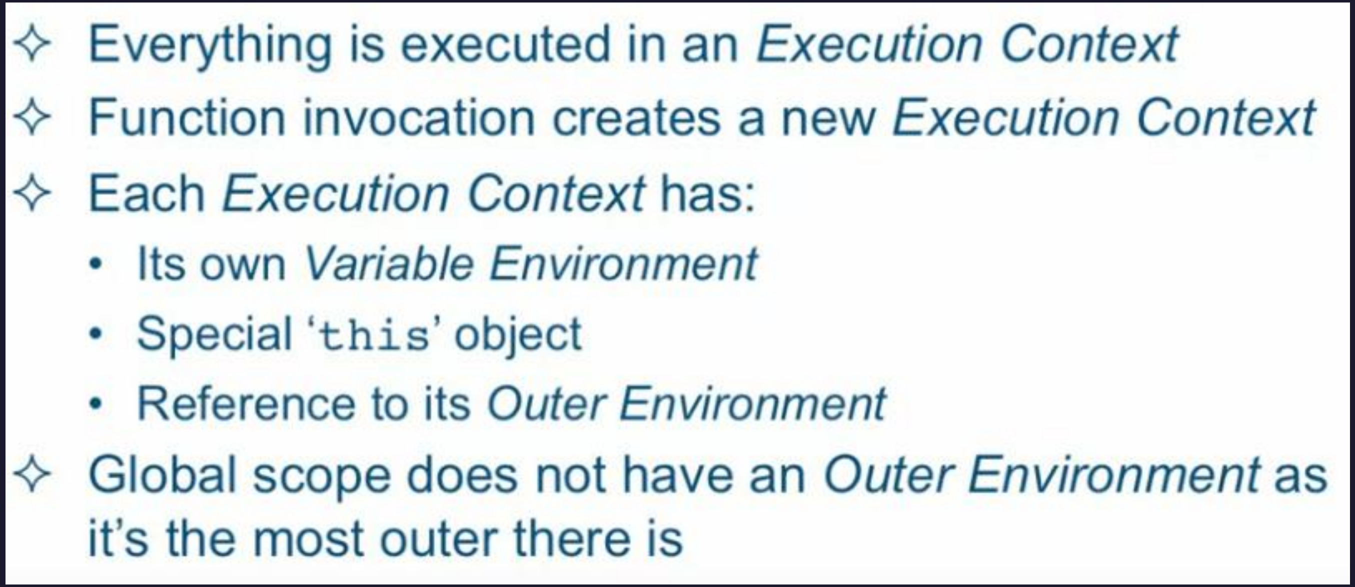
Global

- Variables and functions defined here are available everywhere

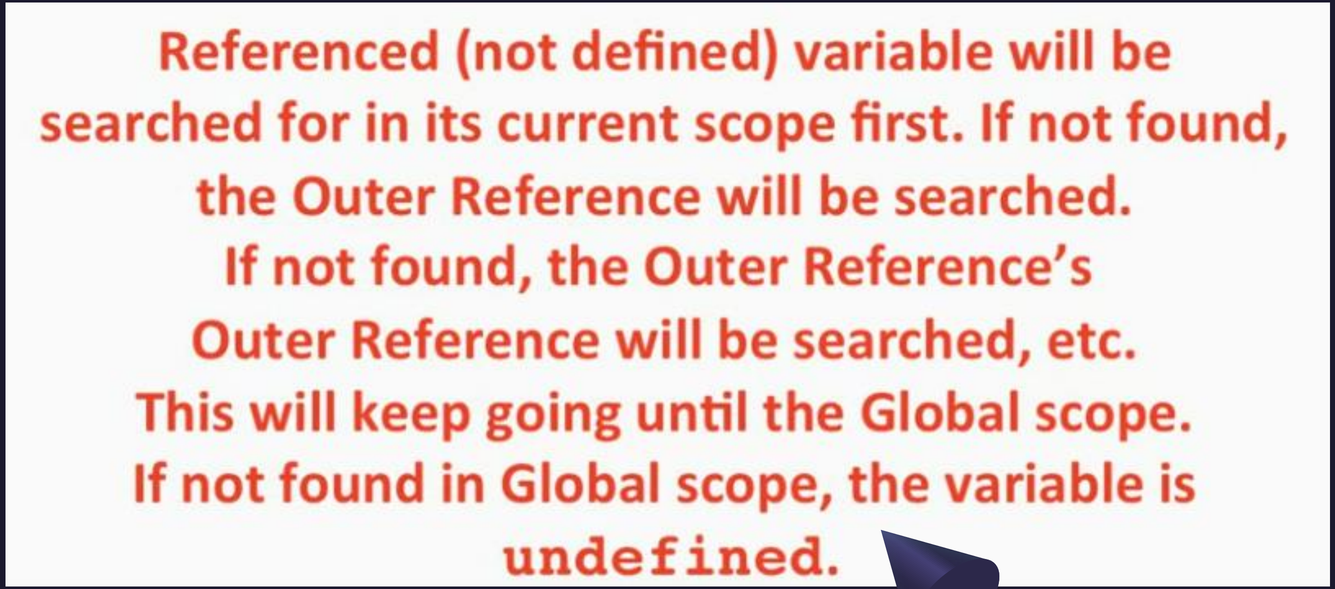
Function
aka lexical

- Variables and functions defined here are available only within this function

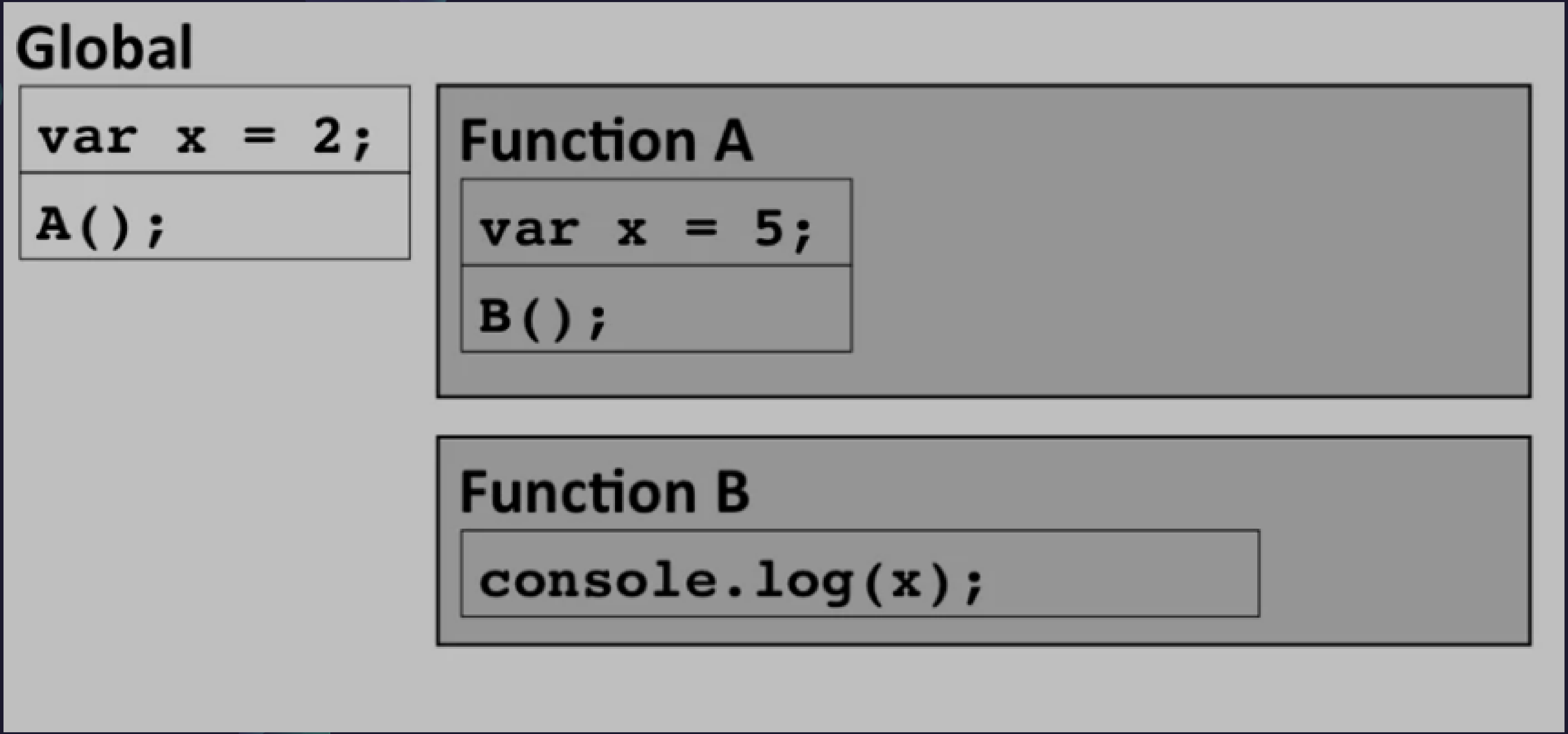
Scope Chain



- ✧ Everything is executed in an *Execution Context*
- ✧ Function invocation creates a new *Execution Context*
- ✧ Each *Execution Context* has:
  - Its own *Variable Environment*
  - Special '`this`' object
  - Reference to its *Outer Environment*
- ✧ Global scope does not have an *Outer Environment* as it's the most outer there is



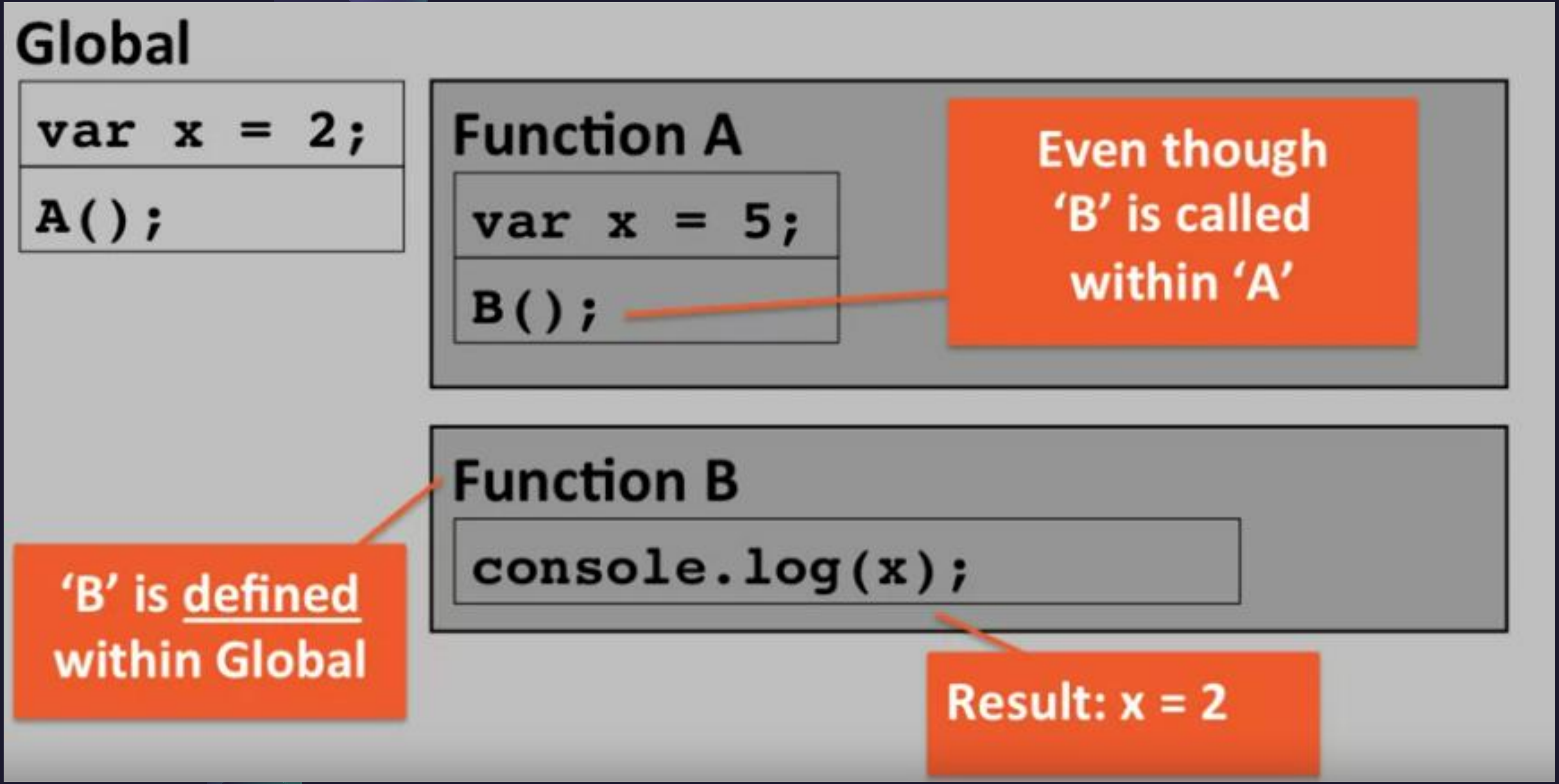
Referenced (not defined) variable will be searched for in its current scope first. If not found, the Outer Reference will be searched. If not found, the Outer Reference's Outer Reference will be searched, etc. This will keep going until the Global scope. If not found in Global scope, the variable is `undefined`.



```
var x = 2;
A();
```

```
var x = 5;
B();
```

```
console.log(x);
```


Global
Function A
Function B
Even though 'B' is called within 'A'
'B' is defined within Global
Result: x = 2

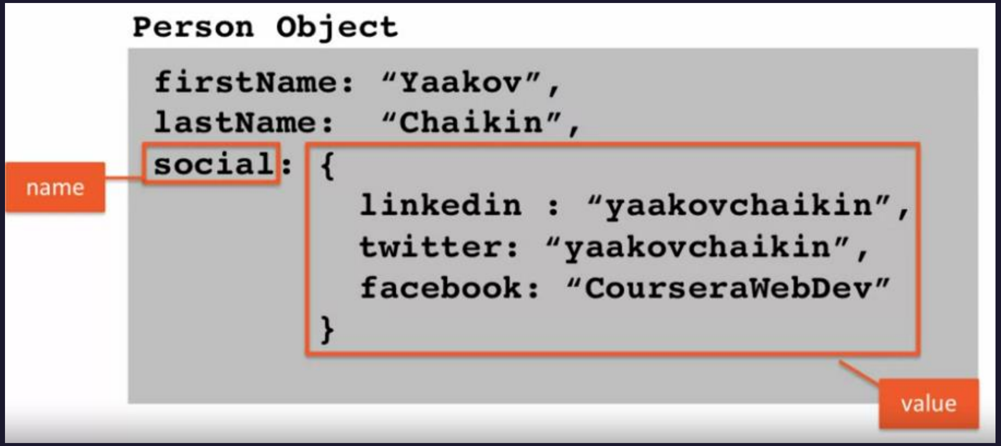
```
var x = 2;
A();
```

```
var x = 5;
B();
```

```
console.log(x);
```

Javascript Types

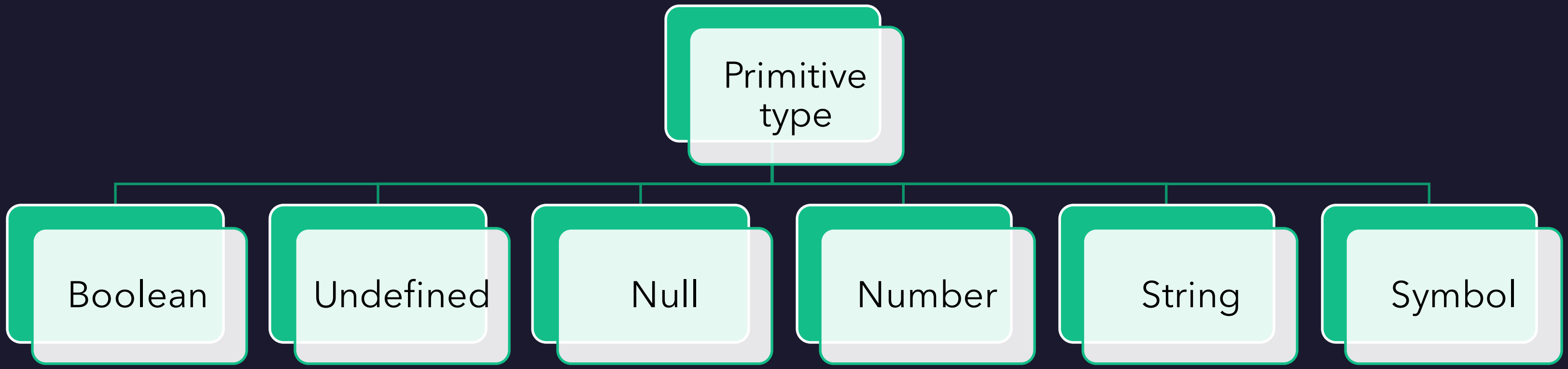
- Language defines its own built-in types.
- JS has 7 built-in types : 6 primitive and 1 object type.
- Object Type : collection of name/value pairs.

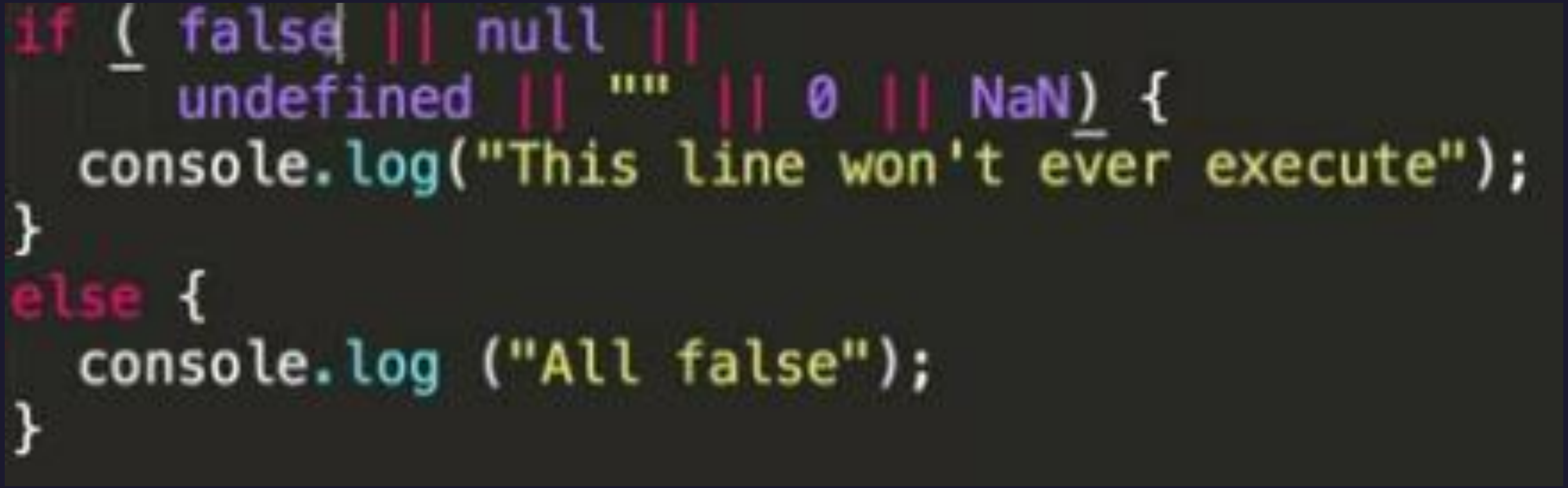

Person Object

```
firstName: "Yaakov",
lastName:  "Chaikin",
social: {
    linkedin : "yaakovchaikin",
    twitter: "yaakovchaikin",
    facebook: "CourseraWebDev"
}
```

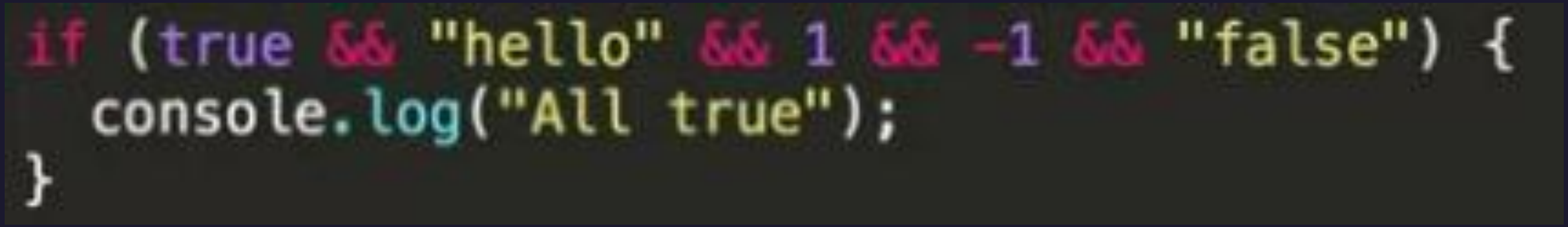
Primitive Type

- Single, immutable value
- **Boolean** : true or false
- **Undefined** : no value has been set.
- **Null** : lack of value but can be set.
- **Number** : numeric type (floating point representation)
- **String** : sequence of characters
- **Symbol** : for private and/or unique keys.





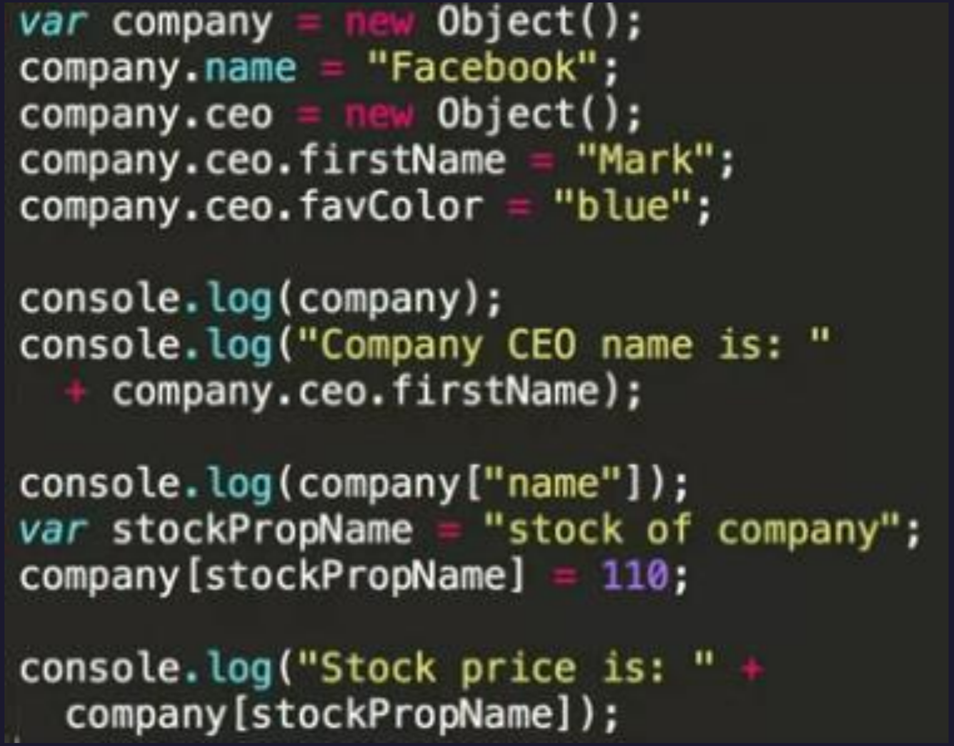
```
if ( false || null ||
     undefined || "" || 0 || NaN) {
  console.log("This line won't ever execute");
}
else {
  console.log ("All false");
}
```



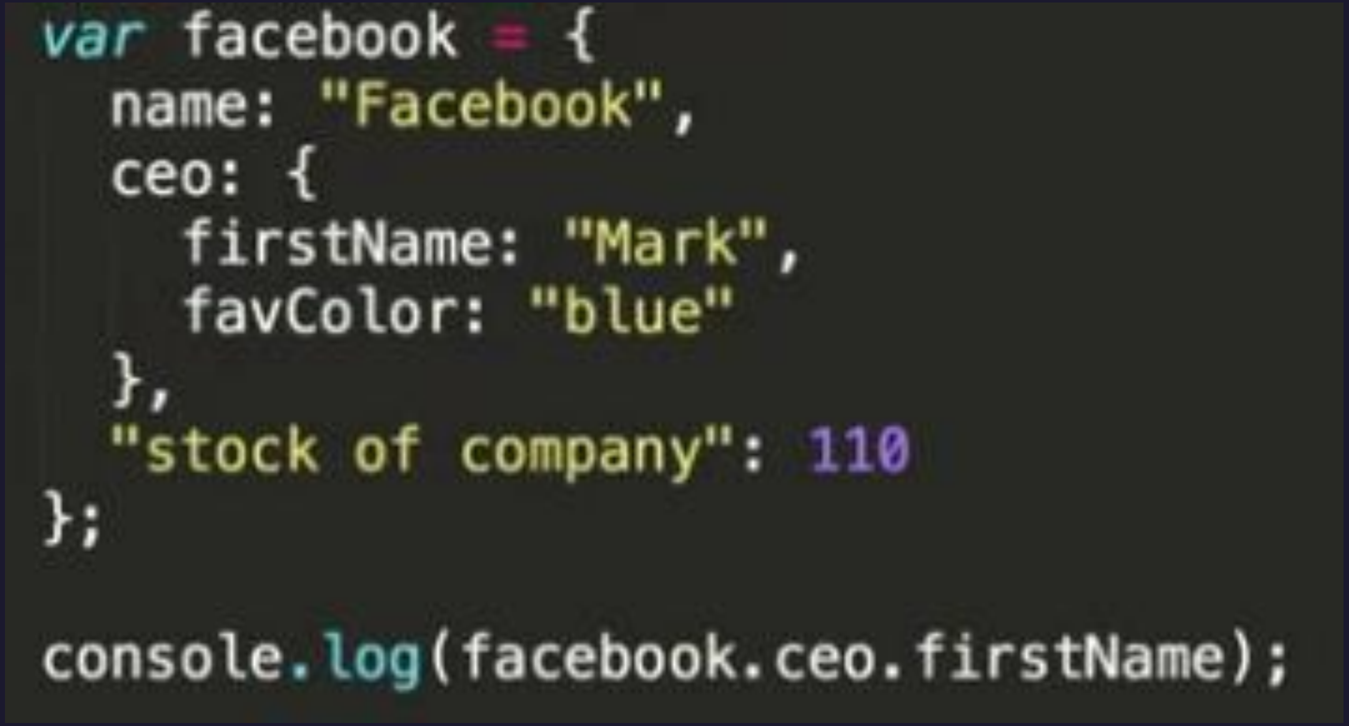
```
if (true && "hello" && 1 && -1 && "false") {
  console.log("All true");
}
```

Object creation

- Using Object literal
- Using Object syntax

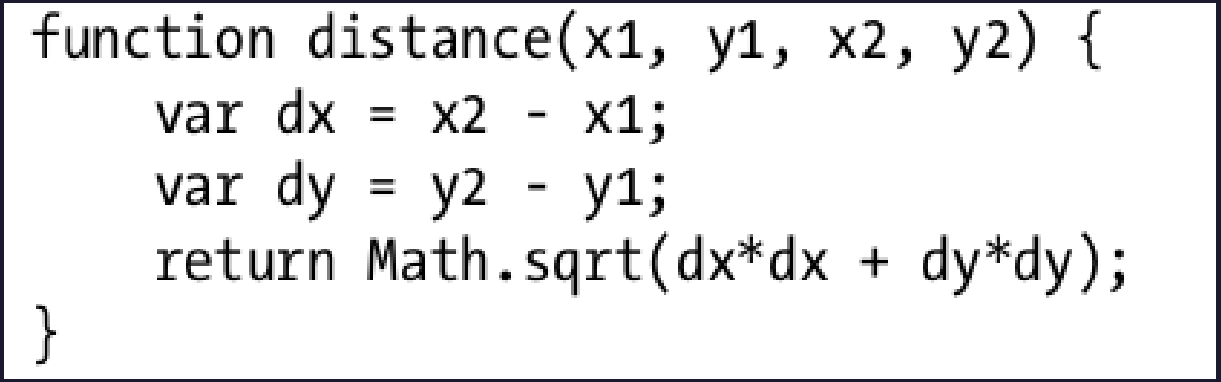


```
var company = new Object();
company.name = "Facebook";
company.ceo = new Object();
company.ceo.firstName = "Mark";
company.ceo.favColor = "blue";

console.log(company);
console.log("Company CEO name is: "
  + company.ceo.firstName);

console.log(company["name"]);
var stockPropName = "stock of company";
company[stockPropName] = 110;

console.log("Stock price is: " +
  company[stockPropName]);
```



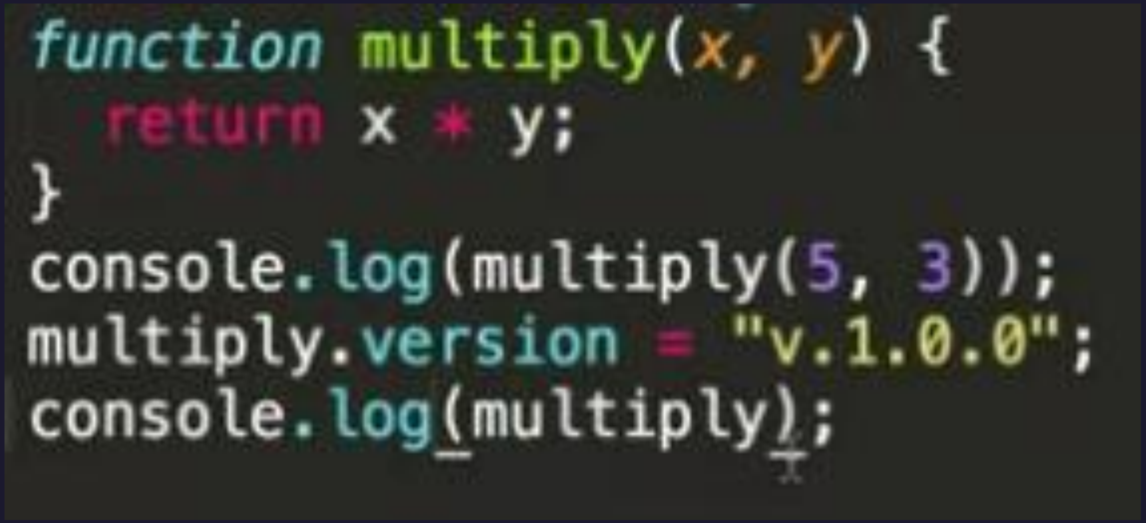
```
var facebook = {
  name: "Facebook",
  ceo: {
    firstName: "Mark",
    favColor: "blue"
  },
  "stock of company": 110
};

console.log(facebook.ceo.firstName);
```

Function

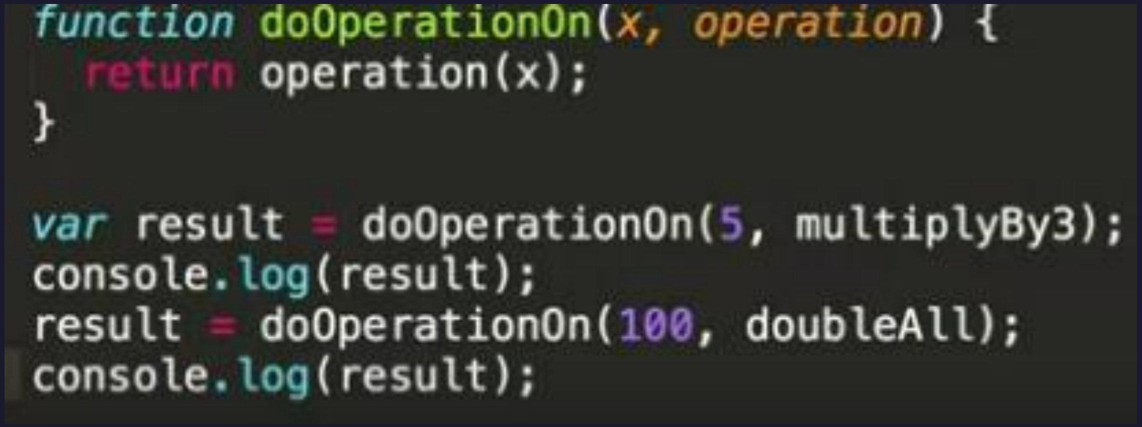


```
function distance(x1, y1, x2, y2) {
    var dx = x2 - x1;
    var dy = y2 - y1;
    return Math.sqrt(dx*dx + dy*dy);
}
```

- A function is a block of JavaScript code that is defined once but may be executed, or invoked, any number of times.
- A function definition may include a list of identifiers, known as *parameters*, that work as local variables for the body of the function.
- Function invocations provide values, or *arguments*, for the function's parameters.



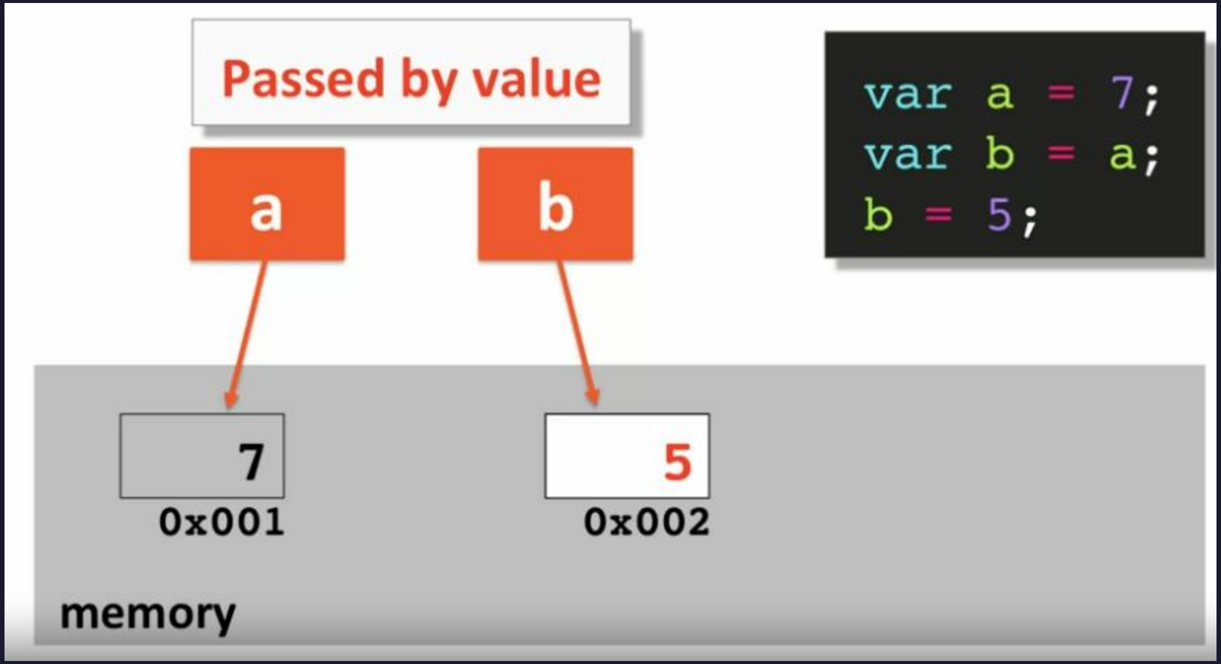
```
function multiply(x, y) {
  return x * y;
}
console.log(multiply(5, 3));
multiply.version = "v.1.0.0";
console.log(multiply);
```



```
function doOperationOn(x, operation) {
  return operation(x);
}

var result = doOperationOn(5, multiplyBy3);
console.log(result);
result = doOperationOn(100, doubleAll);
console.log(result);
```

- Functions are **First class data types** : whatever you could do with the variable, whatever you could do with an object you could also do with the function.
- Functions are *objects*.
- Passing functions as *arguments*.

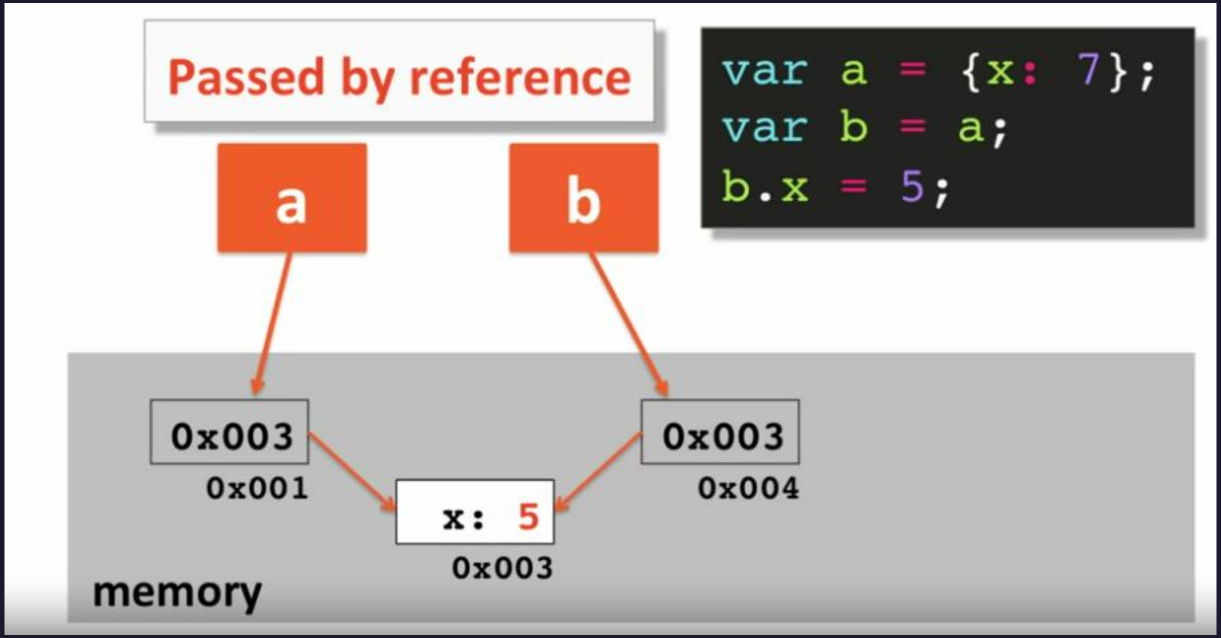
Pass by Value



```
var a = 7;
var b = a;
b = 5;
```

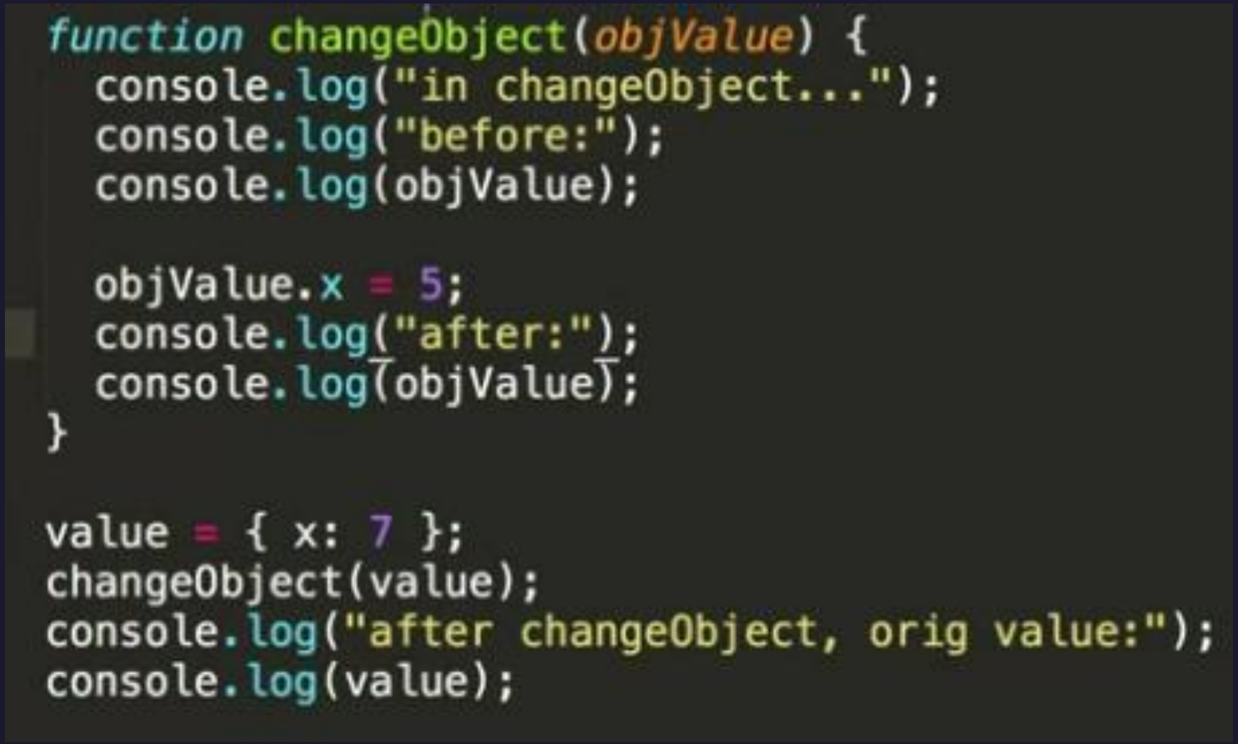
- Given ***b=a***, pass by value means changing copied value in ***b*** does not affect the value in ***a*** and vice-versa.
- Primitives are passed by value and objects are passed by reference.

Pass by Reference

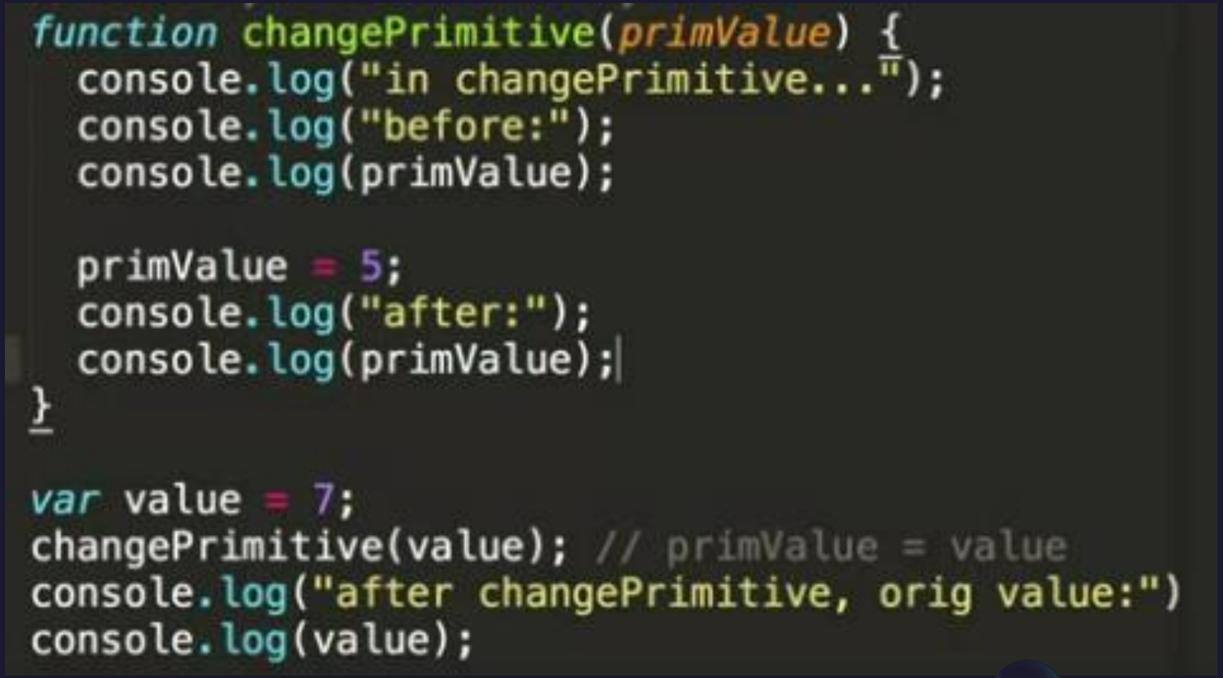


```
var a = {x: 7};
var b = a;
b.x = 5;
```

- Given ***b=a***, pass by reference means changing copied value in ***b*** does affect the value in ***a*** and vice-versa.



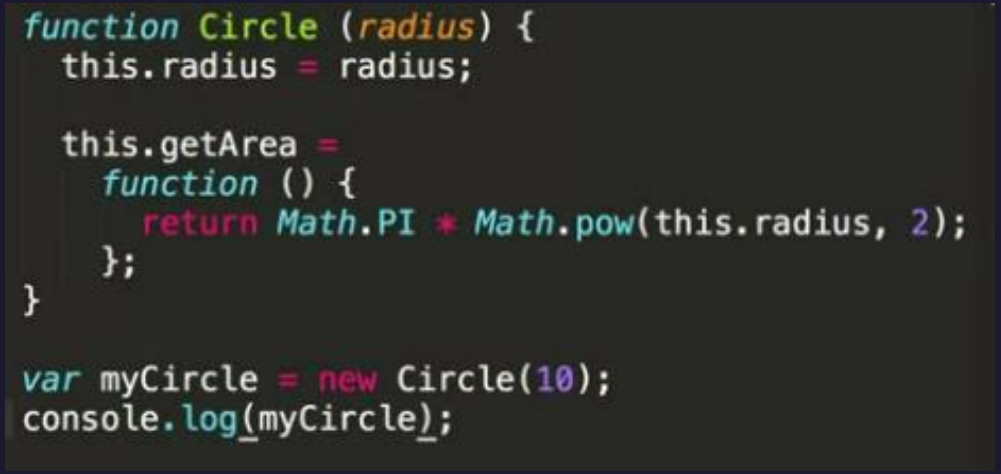
```
function changeObject(objValue) {
  console.log("in changeObject...");
  console.log("before:");
  console.log(objValue);

  objValue.x = 5;
  console.log("after:");
  console.log(objValue);
}

value = { x: 7 };
changeObject(value);
console.log("after changeObject, orig value:");
console.log(value);
```



```
function changePrimitive(primValue) {
  console.log("in changePrimitive...");
  console.log("before:");
  console.log(primValue);

  primValue = 5;
  console.log("after:");
  console.log(primValue);
}

var value = 7;
changePrimitive(value); // primValue = value
console.log("after changePrimitive, orig value:")
console.log(value);
```

Constructors

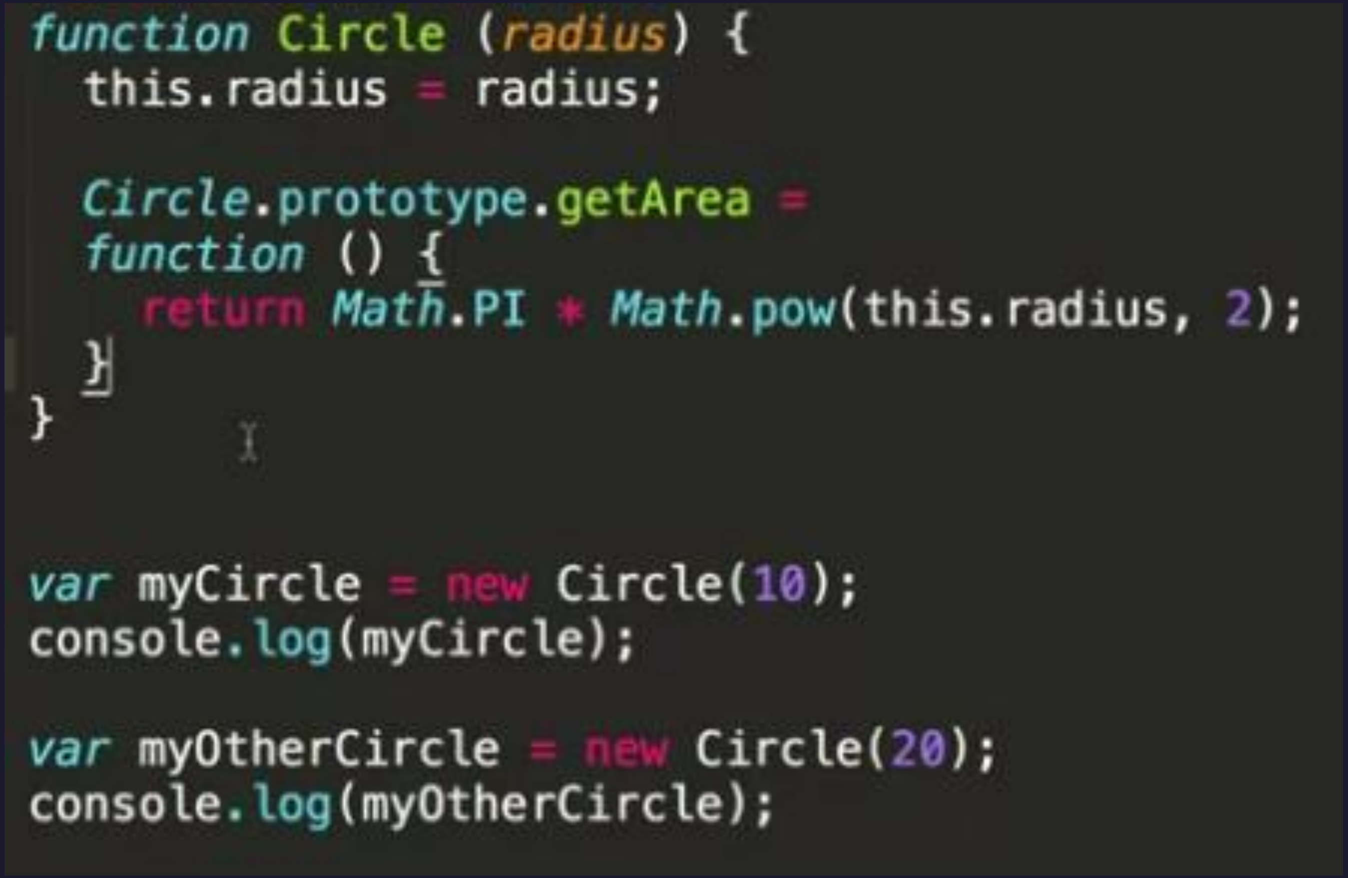
- A constructor is a function designed for the initialization of newly created objects.
- Constructors are invoked using the new keyword.
- The critical feature of constructor invocations is that the prototype property of the constructor is used as the prototype of the new object. This means that all objects created with the same constructor inherit from the same object and are therefore members of the same class.
- We can't return anything from a constructor.



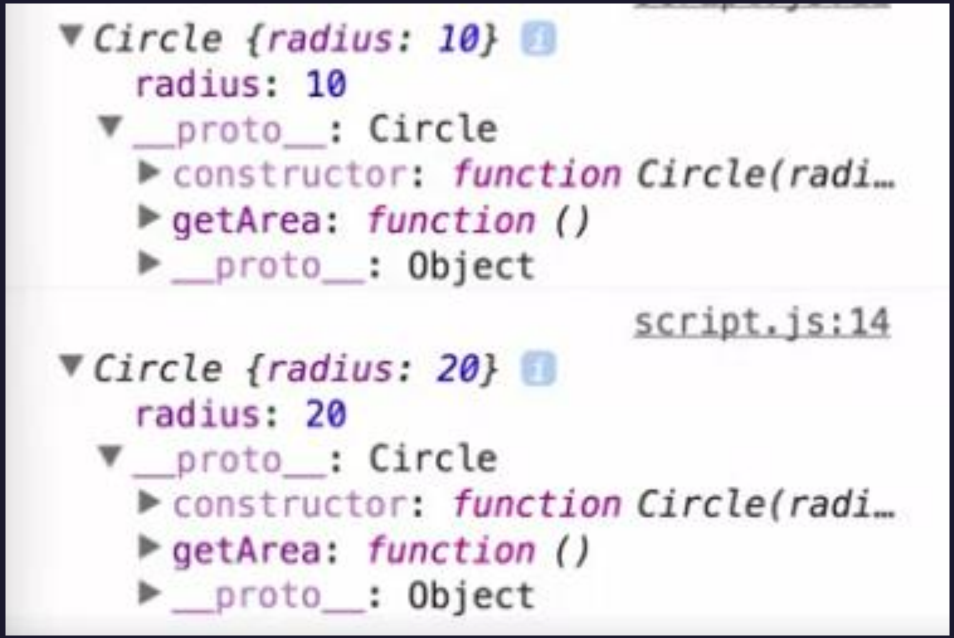
```
function Circle (radius) {
  this.radius = radius;

  this.getArea =
    function () {
      return Math.PI * Math.pow(this.radius, 2);
    };
}

var myCircle = new Circle(10);
console.log(myCircle);
```

Prototype

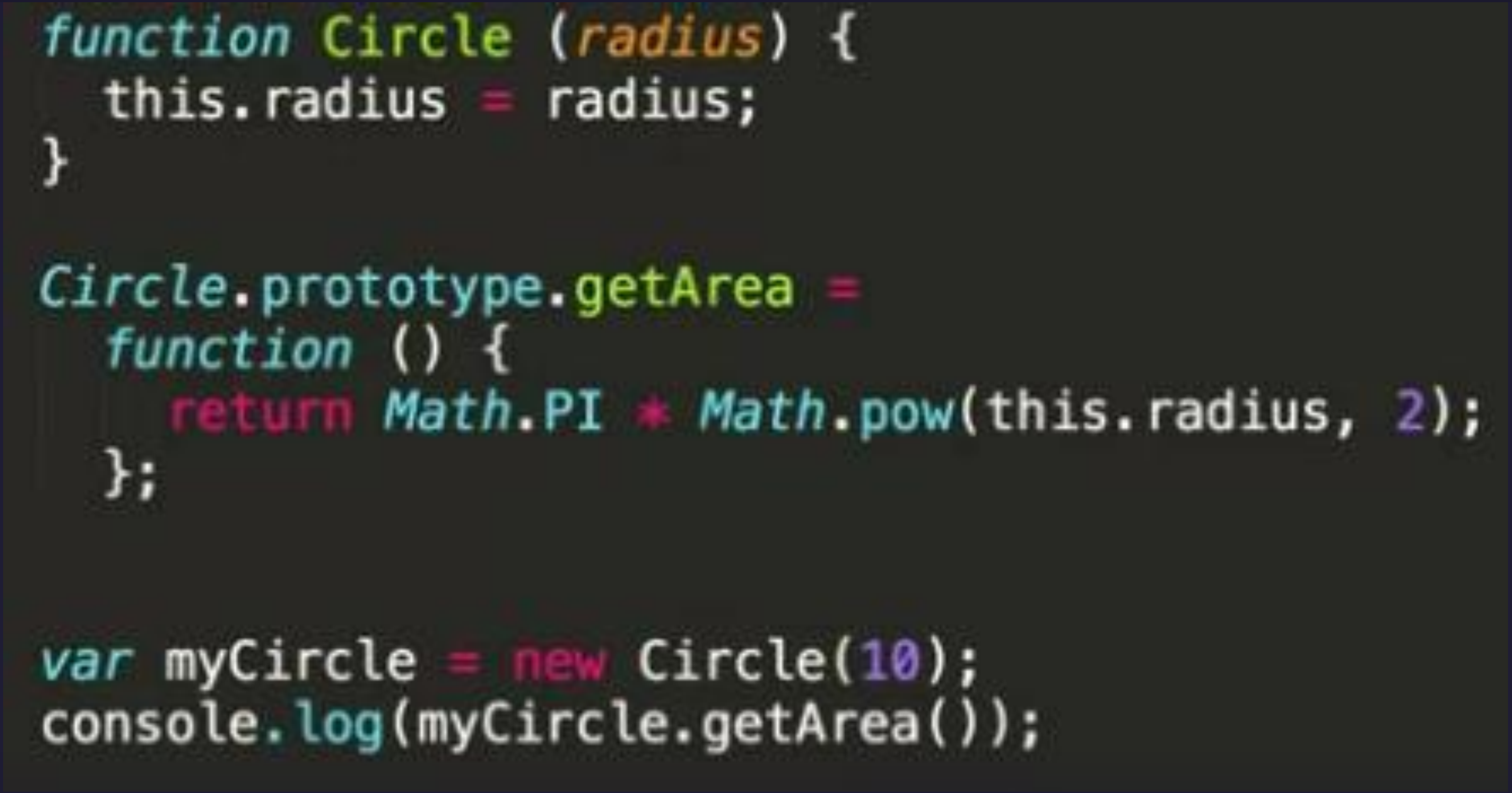
It’s a property of the class
NOT the instance of the class.

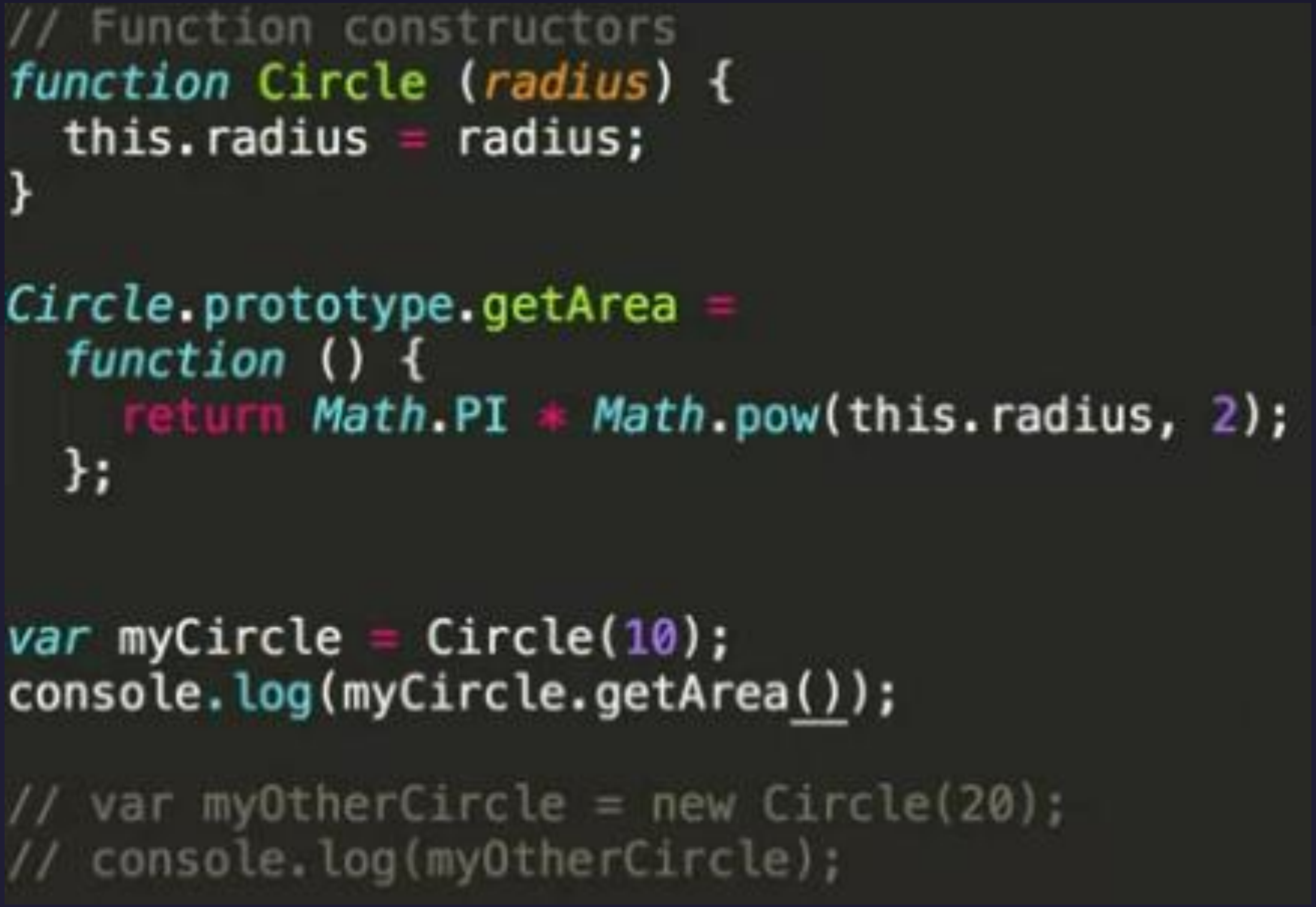


```
function Circle (radius) {
  this.radius = radius;

  Circle.prototype.getArea =
  function () {
    return Math.PI * Math.pow(this.radius, 2);
  }
}


var myCircle = new Circle(10);
console.log(myCircle);

var myOtherCircle = new Circle(20);
console.log(myOtherCircle);
```

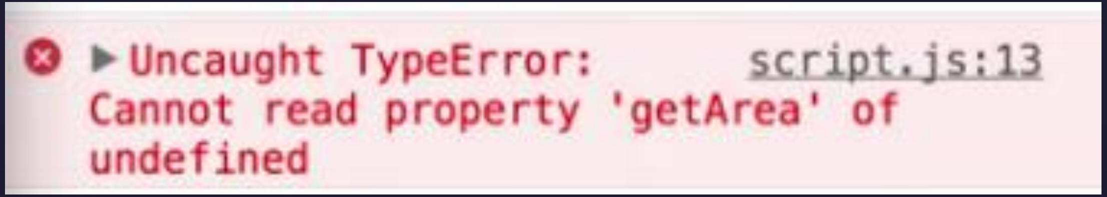


```
▼Circle {radius: 10}
   radius: 10
  ▼__proto__: Circle
   ▶constructor: function Circle(radi…
   ▶getArea: function ()
   ▶__proto__: Object
                                   script.js:14
▼Circle {radius: 20}
   radius: 20
  ▼__proto__: Circle
   ▶constructor: function Circle(radi…
   ▶getArea: function ()
   ▶__proto__: Object
```


```javascript
function Circle (radius) {
  this.radius = radius;
}

Circle.prototype.getArea =
  function () {
    return Math.PI * Math.pow(this.radius, 2);
  };


var myCircle = new Circle(10);
console.log(myCircle.getArea());
```



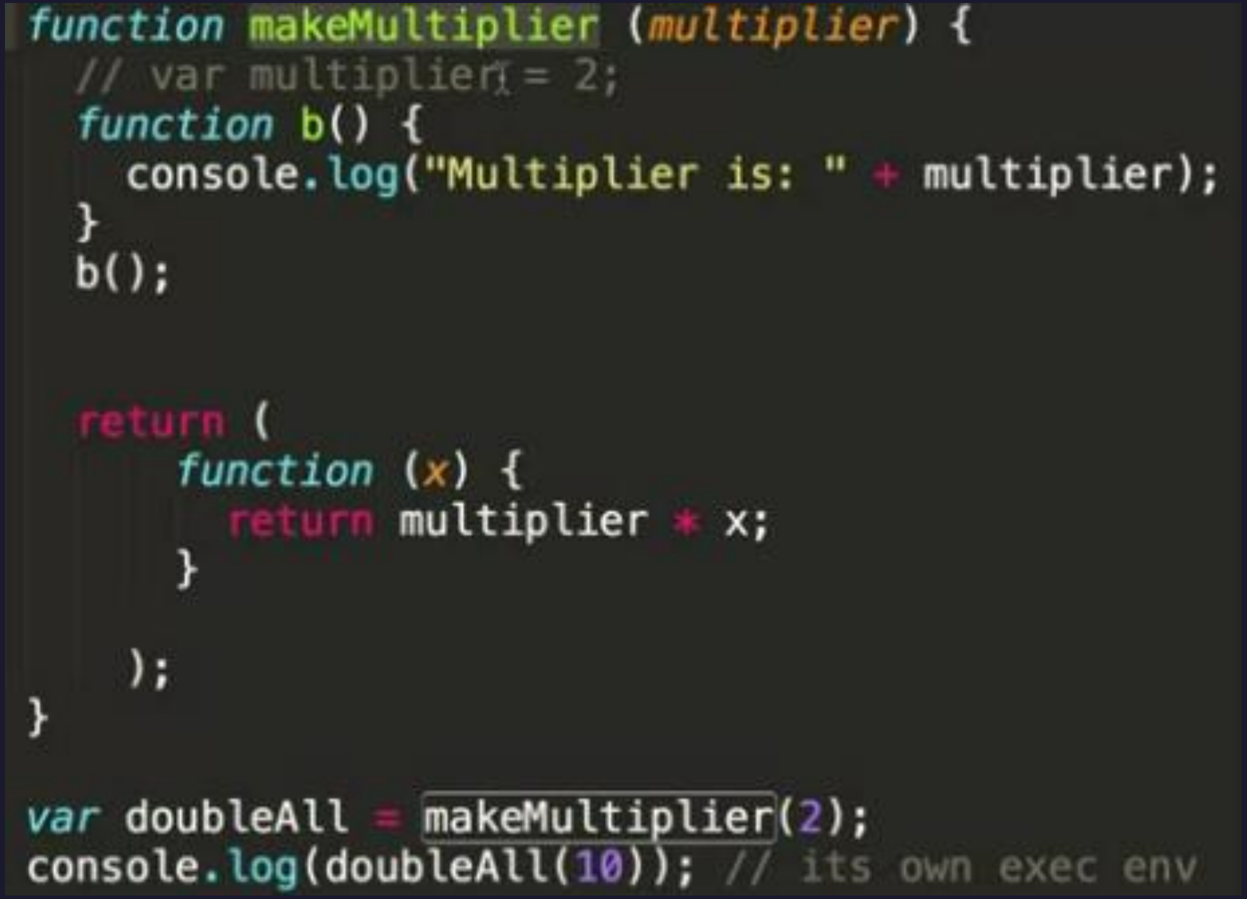
```
// Function constructors
function Circle (radius) {
  this.radius = radius;
}

Circle.prototype.getArea =
  function () {
    return Math.PI * Math.pow(this.radius, 2);
  };


var myCircle = Circle(10);
console.log(myCircle.getArea());

// var myOtherCircle = new Circle(20);
// console.log(myOtherCircle);
```



```
▶Uncaught TypeError:          script.js:13
Cannot read property 'getArea' of
undefined
```

Closure

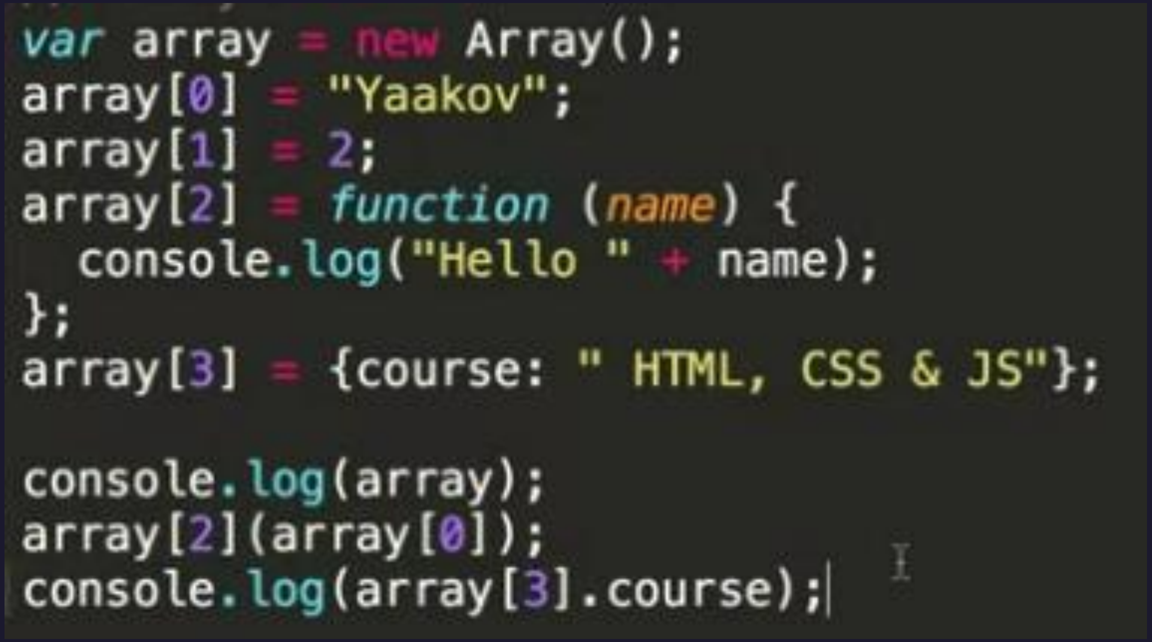
- JavaScript uses ***lexical scoping*** which means that functions are executed using the variable scope that was in effect when they were defined, not the variable scope that is in effect when they are invoked.
- In order to implement lexical scoping, the internal state of a JavaScript function object must include not only the code of the function but also a reference to the current scope chain.
- This combination of a function object and a scope (a set of variable bindings) in which the function's variables are resolved is called a closure.



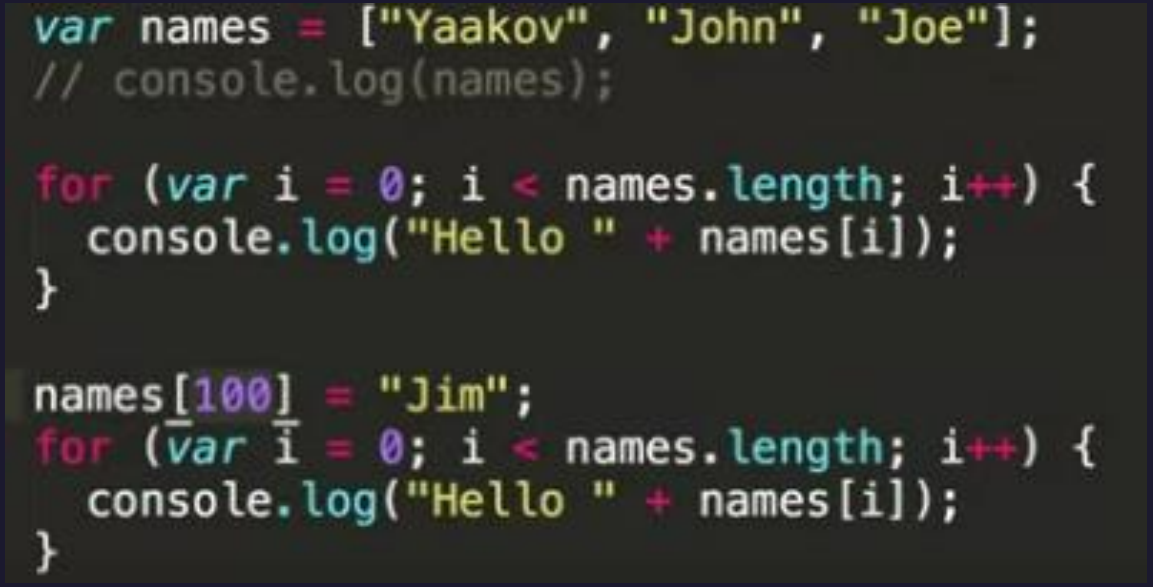
```
function makeMultiplier (multiplier) {
  // var multiplier = 2;
  function b() {
    console.log("Multiplier is: " + multiplier);
  }
  b();


  return (
      function (x) {
        return multiplier * x;
      }

    );
}

var doubleAll = makeMultiplier(2);
console.log(doubleAll(10)); // its own exec env
```

Arrays

- An array is an ordered collection of values. Each value is called an element, and each element has a numeric position in the array, known as its index.
- JavaScript arrays are ***untyped***: an array element may be of any type, and different elements of the same array may be of different types.
- JavaScript arrays are ***zero-based*** and use 32-bit indexes.
- JavaScript arrays are ***dynamic***: they grow or shrink as needed and there is no need to declare a fixed size for the array when you create it or to reallocate it when the size changes.
- JavaScript arrays may be ***sparse***: the elements need not have contiguous indexes and there may be gaps.



```
var array = new Array();
array[0] = "Yaakov";
array[1] = 2;
array[2] = function (name) {
  console.log("Hello " + name);
};
array[3] = {course: " HTML, CSS & JS"};

console.log(array);
array[2](array[0]);
console.log(array[3].course);
```



```
var names = ["Yaakov", "John", "Joe"];
// console.log(names);

for (var i = 0; i < names.length; i++) {
  console.log("Hello " + names[i]);
}

names[100] = "Jim";
for (var i = 0; i < names.length; i++) {
  console.log("Hello " + names[i]);
}
```

Array

Thank you