DOCENTE	Shadi Lahham
Corso	Software Developer
Unità Formativa	Programmazione WEB – Javascript
Argomento	Specificato nel titolo della slide successiva

JS in depth

Objects, arrays, functions and more

Shadi Lahham - Web development

Let, var & const

Let vs var

```
function worker() {
    let x = 88;
    for (let i = 0; i < 4; i++) {
        console.log('i block =', i);
    }
    console.log('x func =', x);
    console.log('i !block =', i); // undefined
}

worker();
console.log('x !func =', x); // undefined
```

let: Block-scoped

Access restricted to nearest enclosing block

```
function worker() {
    var x = 88;
    for (var i = 0; i < 4; i++) {
        console.log('i block =', i);
    }
    console.log('x func =', x);
    console.log('i !block =', i); // output?
}
worker();
console.log('x !func =', x); // undefine
```

var: Function-scoped

Access restricted to nearest enclosing function
Common in older Javascript code

Const

```
let x = 88;
const y = 77;
x = 9;
console.log('x = ', x);
y = 17;  // TypeError: Assignment to constant variable.
console.log('y = ', y);
const y = 55; // SyntaxError: Identifier 'y' has already been declared
```

const:
Block-scoped, like **let**
Values of const variables cannot be reassignment
Const variables cannot be redeclared

Let bug in IE11

```javascript
for (let i = 0; i < 3; ++i) {
    setTimeout(function() {
        console.log(i);
    }, i * 100);
}

// output on chrome 0,1,2
// output on IE11 3,3,3
```

Browser support
Let

Flow

Conditional (Ternary) Operator

```javascript
// example 1
let age = 8;
let voteable = age < 18 ? 'Too young' : 'Old enough';

// example 2
let now = new Date();
let greeting = 'Good' + (now.getHours() > 17 ? ' evening.' : ' day.');
```

Conditional (Ternary) Operator

```
// example 3 - if else vs ternary operator
let x = 71;
let y = 8;
let result;
if (x > y) {
  result = 'good job';
} else {
  result = 20;
}
// Below is the above code written using the ternary operator
let result2 = x > y ? 'good job' : 20;
```

Conditional (Ternary) Operator

```
// chaining
let bar;
let foo = bar === 'a' || bar === 'b' ? (bar === 'a' ? 1 : 2) : 3;

// is this too much??
let i = 5;
let result = i % 2 == 0 ? 'a' : i % 3 == 0 ? 'b' : i % 5 == 0 ? 'c' : i % 7 == 0 ? 'd' : 'e';
```

Arrays

Access and assignment

```
let teachers = ['Gina', 'Amanda', 'Brenda', 'Amy'];
let classes = [];

classes[0] = 'HTML';
classes[1] = 'CSS';
classes.push('JS');
classes.pop();

let i = 0;
classes[i];
classes[1];
classes.pop();
classes.length;
```

Iteration

```javascript
let classes = [];
classes[0] = 'HTML';
classes[1] = 'CSS';
classes.push('JS');
for (let i = 0; i < classes.length; i++) {
  console.log(i);
}
```

For .. of

```
let countries = ['Italy', 'France', 'Germany'];

for (let country of countries) {
  console.log(country);
}
```

Browser support
for...of

ForEach

```javascript
let numbers = [1, 2, 3, 4];

// using for
for (let i = 0; i < numbers.length; i++) {
  console.log(numbers[i]);
}

// using forEach
numbers.forEach(function (number) {
  console.log(number);
});
```

Map

```
let numbers = [1, 2, 3, 4];

// using for
let newNumbers = [];
for (let i = 0; i < numbers.length; i++) {
  newNumbers[i] = numbers[i] * 2;
}

// using map
let newNumbers2 = numbers.map(function (number) {
  return number * 2;
});
```

Method chaining

```javascript
let numbers = [1, 2, 3, 4];
let newNumbers = numbers
  .map(function (number) {
    return number * 2;
  })
  .map(function (number) {
    return number + 1;
  });



// shorter version
let numbers = [1, 2, 3, 4];
let newNumbers = numbers.map(number => number * 2).map(number => number + 1);
```

Filter

```javascript
let numbers = [1, 2, 3, 4];

// using for
let newNumbers = [];
for (let i = 0; i < numbers.length; i++) {
  if (numbers[i] % 2 !== 0) {
    newNumbers.push(numbers[i] * 2);
  }
}

// using filter
let newNumbers2 = numbers
  .filter(function (number) {
    return number % 2 !== 0;
  })
  .map(function (number) {
    return number * 2;
  });
```

Reduce

```
let numbers = [1, 2, 3, 4];

// using for
let totalNumber = 0;
for (let i = 0; i < numbers.length; i++) {
  totalNumber += numbers[i] * 2;
}

// using reduce
let totalNumber2 = numbers
  .map(function (number) {
    return number * 2;
  })
  .reduce(function (total, number) {
    return total + number;
  }, 0);
```

Reduce

```
let some = [1, 2, 3, 4, 5, 6, 7].reduce(function (accu, curr) {
  if (Math.random() > 0.5) {
    accu.push(curr);
  }
  return accu;
}, []);
console.log(some);
```

Reduce

```javascript
let kebab = ['I', 'hAve', 'A', 'drEam'].reduce(function (accu, curr, index, arr) {
  const word = index === arr.length - 1 ? curr : curr + '-';
  return accu + word.toLowerCase();
}, '');
console.log(kebab);
```

Objects

Creation and assignment

```javascript
// create object and assign property
let cat = {};
cat.furColor = 'orange';

// create and assign
let cat2 = { furColor: 'orange' };

// Object literal
let cat = {
  age: 5,
  furColor: 'orange',
  isHappy: true,
  likes: ['sleep', 'milk'],
  birthday: { month: 7, day: 17, year: 2020 }
};
```

Nested objects

```javascript
let doll = {
  size: 'large',
  innerDoll: { size: 'medium' }
};
doll.innerDoll.innerDoll = { size: 'small' };

console.log(doll);
```

Array of objects

```
// array of objects
let cats = [
  { name: 'Angel', age: 18, furColor: 'grey' },
  { name: 'Evil', age: 14, furColor: 'red' },
  { name: 'Meh', age: 12, 'Fur Color': 'white' }
];
console.log(cats);
console.log(cats[1].furColor);
console.log(cats[2]['Fur Color']);
```

Dot notation

```javascript
// Dot notation

// reading properties
let furVariable = cat.furColor;
console.log(furVariable);

// modifying properties
cat.furColor = 'grey';
console.log(furVariable);
```

Bracket notation

```
// cannot be done with dot notation
cat['fur color'] = 'orange';
let facebookFriends = {};
facebookFriends[12323] = cat;

// bracket notation with variables
let cat = {};
let prop = 'furColor';
cat[prop] = 'orange';
let color = cat[prop];
```

Bracket notation with variables

```javascript
let socials = ['instagram', 'tiktok', 'twitter', 'pinterest'];

const handshake = {
  pinterestShare: function () {
    // code to share on pinterest
  },
  twitterShare: function () {
    // code to share on twitter
  }
};

for (const social of socials) {
  const callback = handshake[social + 'Share'];
  if ('function' === typeof callback) {
    callback(); // equivalent to handshake[social + 'Share']();
  }
}
```

Iterating using for .. in

```javascript
let zoo = {
  birds: 3,
  bears: 5,
  cats: 12
};
for (let key in zoo) {
  if (zoo.hasOwnProperty(key)) {
    console.log('zoo.' + key + ' = ' + zoo[key]);
  }
}

// remember: for .. in is for objects, for .. of is for arrays
// don't use for..of on objects
const sam = { name: 'sam', age: 42 };
for (const property of sam) {
  // TypeError: sam is not iterable
  console.log(property);
}
```

Object keys(), values() & entries()

```javascript
const zooAnimals = {
  animal1: 'Lion',
  animal2: 'Elephant'
};

Object.keys(zooAnimals).forEach(key => { // keys() returns an array of object's properties
  console.log(key, zooAnimals[key]); // animal1 Lion, animal2 Elephant
});

Object.values(zooAnimals).forEach(val => { // values() returns an array of object's values
  console.log(val); // Lion, Elephant
});

Object.entries(zooAnimals).forEach(entry => { // entries() returns an array of key-value pairs
  const [key, value] = entry;
  console.log(key, value); // animal1 Lion, animal2 Elephant
});
```

Operators

Logical OR ||

The OR || operator using non-boolean values

1. Evaluates operands from left to right
2. For each operand, if it is truthy, stops and returns the original value of the operand
3. If all operands are falsy, returns the last operand

```
let name = '';
let userName = name || 'default'; // default

let name2 = 'james';
let userName2 = name2 || 'default'; // james
```

Logical OR ||

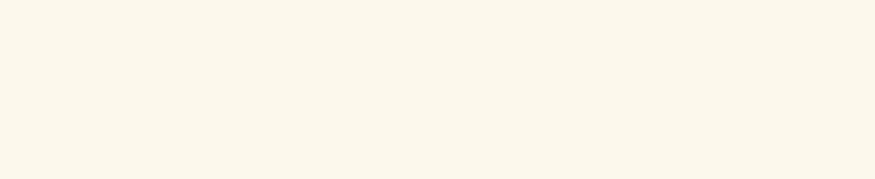# Logical AND &&

The AND && operator using non-boolean values

1. Evaluates operands from left to right
2. For each operand, if it is falsy, stops and returns the original value of that operand
3. If all operands are truthy, returns the last operand

```
let userName = person && person.name; // undefined


let person = {};
userName = person && person.name; // undefined


person = { name: 'james' };
userName = person && person.name; // james
```

[Logical AND &&](#)

Functions

Another way to look at functions

```
let add = function(a, b) {
  return a + b;
};

let mad = add;

let resultA = add(5, 4); // 9

let resultB = mad(21, 7); // 28

console.log(typeof add); // function
```

// **note:** functions are regular objects with the additional capability of being invokable

Another way to look at functions

```javascript
function add(a, b) {
  return a + b;
}

let mult = function(a, b) {
  return a * b;
};

let calculate = function(fn, a, b) {
  console.log('This is your result:', fn(a, b));
};

calculate(add, 2, 4);
calculate(mult, 2, 4);


// note: functions can be passed as parameters
```

Return and side effects

```javascript
let greeter = function(name, place) {
  return 'Mister ' + name + ' of' + place;
};

// function with a side effect
let nameLogger = function(name, place) {
  let newName = 'Mister ' + name + ' of' + place;
  console.log(newName);
  return newName;
};
```

Function arguments

```
let add = function(a, b) {
  console.log(arguments); // logs [3,10]
  return a + b;
};

let sum = add(3, 10); // 13
```

Function arguments

```javascript
let addMany = function() {
  let sum = 0;
  for (let i = 0; i < arguments.length; i++) {
    sum += arguments[i];
  }
  return sum;
};

let sumA = addMany(3, 10, 57, 24); // 94
let sumB = addMany(3, 10, 57, 24, 200, 300); // 594

// implement a function that returns the max of n arguments; your own version of Math.max()
```

Default arguments

```javascript
let nameLogger = function (name, adj) {
  if (adj === undefined) {
    adj = 'wonderful';
  }
  let newName = 'The ' + adj + ' Mr.' + name;
  console.log(newName);
};

nameLogger('adam', 'lazy');
nameLogger('james');
```

Global and local precedence

```
let g = 'global';

function go() {
  let l = 'local';
  let g = 'in here!';
  console.log(g + ' inside go');
}

go();
console.log(g + ' outside go');
```

Your turn

1.Soundwave

Given the following array
```
let noisesArray = ['quack', 'sneeze', 'boom'];
```

Produce the following array, then print it to the console
```
['Quack!','qUack!!','quAck!!!','quaCk!!!!','quacK!!!!!','Sneeze!','sNeeze!!','snEeze!!!','sneEze!!!!
','sneeZe!!!!!','sneezE!!!!!!','Boom!','bOom!!','boOm!!!','booM!!!!']
```

2.Babies

- Create an empty array of babies
- Each baby should have the following properties
  - "name" (a string)
  - "months" (age in months as number)
  - "noises" (an array of strings)
  - "favoriteFoods" (an array of strings)
- Add 4 different babies to the array using as many different ways as possible
- Iterate through the array printing key and value pairs e.g [name:"Lyla"]
- Now add an "outfit" property to each baby in the array
  - Outfit should describes at least 3 parts of their clothing using different properties, for example, "shirt": "blue"
  - Print each baby again with their outfit in a nicely formatted object

3.Baby processing

Using the babies array from the previous exercise:

- Write a getBabyOutfit() function that returns a description a baby's outfit
  - e.g "Lyla is wearing a blue shirt and red pants and a green hat"
- Write a feedBaby() function that prints what a baby is eating.
  - e.g. "Lyla is eating food3, food1, food4 and food2"
  - All foods in favoriteFoods should appear but randomly each time the function is called
- Run both function on all the babies

4.Clone

Write a function clone() that clones any object
- Test it on the object in the next slide
- Change the name of the cloned object and make sure that the original did not change

Important:
Write the function yourself, do not use built-in functions such as Object.assign(), jQuery.extend() or JSON.parse(JSON.stringify())

4.Clone

```
The object to clone and test:
{
  name: 'Green Mueller',
  email: 'Rigoberto_Muller47@yahoo.com',
  address: '575 Aiden Forks',
  bio: 'Tenetur voluptatem odit labore et voluptatem vel qui placeat sit.',
  active: false,
  salary: 37993,
  birth: Sun Apr 18 1965 13:38:00 GMT+0200 (W. Europe Daylight Time),
  bankInformation:
   { amount: '802.04',
     date: Thu Feb 02 2012 00:00:00 GMT+0100 (W. Europe Standard Time),
     business: 'Bernhard, Kuhn and Stehr',
     name: 'Investment Account 8624',
     type: 'payment',
     account: '34889694' }
}
```

Bonus

5.Clone strings

Write a function cloneStrings() that only clones string properties of an object
Starting with the example object of the previous exercise this should be the result

```
{ name: 'Green Mueller',
  email: 'Rigoberto_Muller47@yahoo.com',
  address: '575 Aiden Forks',
  bio: 'Tenetur voluptatem odit labore et voluptatem vel qui placeat sit.',
  bankInformation:
   { amount: '802.04',
     business: 'Bernhard, Kuhn and Stehr',
     name: 'Investment Account 8624',
     type: 'payment',
     account: '34889694' }
}
```

6.Reduce All

Make sure that you fully understand the Array reduce method

Write functions that use the reduce method to implement your version of the following Array methods:
forEach() , map(), filter() , indexOf() , slice()

For each method, implement parameters and return values as in the documentation
- do not use Array.prototype
- your functions receive as a first parameter the array on which to operate
- all other parameters should be identical to the documentation
- except for the thisArg parameter, you don't have to implement it

For example your implementation of forEach could be something like this:
```
function myForEach(arr, ... ) {
}
```

Note: *This exercise is harder than the ones you have done so far. Dedicate enough time for it*

6.Reduce All

Testing:
- write tests that compare the output of your functions to those of the Array methods
- write several and comprehensive tests for each method
- make sure that your methods give the same output as the originals

Note: *See the following slide with an example of how to test myMap()*

6.Reduce All

```javascript
// Example of testing myMap
// group of arrays used for testing
let testGroup = [
    [ 1, 2, 3, 4, 5 ],
    [ 0, 0, 3, 4, 5 ],
    [ 7, 0, 9, 74, 85, 1, 42, 3, 88 ]
];

// test function for testing map - can be any function as long as the parameters are what map
expects
let testFunc = function(num) {
  return num * 2;
};

// replace this with your implementation of map using reduce
function myMap(arr, ... ) {
}
```

6.Reduce All

```javascript
console.log('==== Testing Array.map() method ====');
testGroup.forEach(function(arr) {
  console.log(arr.map(testFunc));
});

console.log('\n==== Testing the function myMap() ====');
testGroup.forEach(function(arr) {
  console.log(myMap(arr, testFunc));
});

// note that tests for forEach, indexOf, filter and slice will be different because the methods
behave differently
```

References

Ternary operator

[Ternary operators vs if-else statements](#)

[Alternative to if / else and switch](#)

[Nested ternary operators](#)

[Ternary operator discussion](#)

[MDN | ternary operator](#)

References

Array methods

[Javascript array reference](#)

[Array - javascript](#)

[Functional programming in javascript: map, filter and reduce](#)

References

for .. in and for .. of

[for...in - JavaScript](#)

[for...of - JavaScript](#)

[for..in versus for..of loops](#)

Logical operators

[The && and || operators in javascript](#)

References

Functions have more complexity than you think

Explore these references to learn more

Javascript function definitions

Every possible way to define a javascript function

Javascript essentials: exploring all the different ways to declare functions

© Copyright & Attribution

Unless otherwise stated, all materials are © 2017–2025 **Shadi Lahham**