Introduction to

JavaScript

the programming language of the Web

Eman Fathi

Arrays

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 **un-typed**: an array element may be of any type, and different elements of the same array may be of different types.
- Array elements may even be objects or other arrays, which allows you to create complex data structures, such as arrays of objects and arrays of arrays.
- JavaScript arrays are **zero-based** and use 32-bit indexes: the index of the first element is 0, and the highest possible index is 4,294,967,295.
- 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.
- Every JavaScript array has a **length** property. this property specifies the number of elements in the array.

Creating Arrays

Array literal syntax

```
var empty = [];  An array with no elements
var primes = [2, 3, 5, 7, 11];  An array with 5 numeric elements
var misc = [ 1.1, true, "a", ];  3 elements of various types + trailing comma
```

Array() constructor

```
var a = new Array();       Equal to []
var a = new Array(10);
var a = new Array(5, 4, 3, 2, 1, "testing, testing");
```

Reading and Writing Array Elements

```
var a = ["world"];  Start with a one-element array
var value = a[0];  Read element 0
a[1] = 3.14;  Write element 1
i = 2;
a[i] = 3;  Write element 2
a[i + 1] = "hello";  Write element 3
a[a[i]] = a[0];  Read elements 0 and 2, write element 3
a[1000] = 0;  Assignment adds one element but sets length to 1001.
```

Array Length

```
[].length  => 0: the array has no elements
['a','b','c'].length  => 3: highest index is 2, length is 3

a = [1,2,3,4,5];  Start with a 5-element array.
a.length = 3;  a is now [1,2,3].
a.length = 0;  Delete all elements. a is [].
a.length = 5;  Length is 5, but no elements, like new Array(5)
```

Adding and Deleting Array Elements

```
a = []  Start with an empty array.
a[0] = "zero";  And add elements to it.
a[1] = "one";

a = [];  Start with an empty array
a.push("zero")  Add a value at the end. a = ["zero"]
a.push("one", "two")  Add two more values. a = ["zero", "one", "two"]

a = [1,2,3];
delete a[1];  a now has no element at index 1
1 in a  => false: no array index 1 is defined
a.length  => 3: delete does not affect array length
```

Iterating Arrays

```
var a = [1,2,3,,5,6];
a[10] = 10;

for(var i = 0; i < a.length; i++) {
    if (!a[i]) continue;  Skip null, undefined, and nonexistent elements
        console.log(a[i]);
}

for(i in a)
    console.log(a[i]);
```

Array.foreach

The forEach() method executes a provided function once per array element.

Syntax :

```
arr.forEach(callback
```

Parameters :

Callback : Function to execute for each element, taking three arguments:

1. currentValue : The current element being processed in the array.
2. Index : The index of the current element being processed in the array.
3. Array : The array filter was called upon.

Return value :

undefined.

forEach() executes the provided callback once for each element present in the array in ascending order. It is not invoked for index properties that have been deleted or are uninitialized (i.e. on sparse arrays).

Array.foreach

There is no way to stop or break a forEach() loop other than by throwing an exception. If you need such behavior, the forEach() method is the wrong tool, use a plain loop instead. If you are testing the array elements for a predicate and need a Boolean return value, you can use every() or some() instead.

Examples : Printing the contents of an array

The following code logs a line for each element in an array:

```
function logArrayElements(element, index, array) {
    console.log('a[' + index + '] = ' + element);
}
// Notice that index 2 is skipped since there is no item at that position in the array.
[2, 5, , 9].forEach(logArrayElements);
// a[0] = 2
// a[1] = 5
// a[3] = 9
```

Array.some

The some() method tests whether some element in the array passes the test implemented by the provided function.

Syntax :

```
arr.some(callback)
```

Parameters :

Callback : Function to execute for each element, taking three arguments:

1. currentValue : The current element being processed in the array.
2. Index : The index of the current element being processed in the array.
3. Array : The array filter was called upon.

Return value :

true if the callback function returns a truthy value for any array element; otherwise, false.

some() executes the callback function once for each element present in the array until it finds one where callback returns a truthy value (a value that becomes true when converted to a Boolean). If such an element is found, some() immediately returns true. Otherwise, some() returns false.

Array.some

Examples : Testing value of array elements

The following example tests whether any element in the array is bigger than 10.

```
function isBiggerThan10(element, index, array) {
    return element > 10;
}
[2, 5, 8, 1, 4].some(isBiggerThan10);  // false
[12, 5, 8, 1, 4].some(isBiggerThan10); // true
```

Array.every

The every() method tests whether all elements in the array pass the test implemented by the provided function.

Syntax :

```
arr.every(callback)
```

Parameters :

Callback : Function to execute for each element, taking three arguments:

1. currentValue : The current element being processed in the array.
2. Index : The index of the current element being processed in the array.
3. Array : The array filter was called upon.

Return value :

true if the callback function returns a truthy value for every element; otherwise, false.

The every method executes the provided callback function once for each element in the array until it finds one where callback returns a falsy value (a value that becomes false when converted to a Boolean). If such an element is found, the every method immediately returns false. Otherwise, if callback returned a true value for all elements, every will return true.

Array.every

Examples : Testing size of all array elements

The following example tests whether all elements in the array are bigger than 10.

```
function isBigEnough(element, index, array) {
    return element >= 10;
}
[12, 5, 8, 130, 44].every(isBigEnough);   // false
[12, 54, 18, 130, 44].every(isBigEnough); // true
```

Array Methods

Array.join() method converts all the elements of an array to strings and concatenates them, returning the resulting string. You can specify an optional string that separates the elements in the resulting string. If no separator string is specified, a comma is used.

```
var a = [1, 2, 3];  Create a new array with these three elements
a.join();  => "1,2,3"
a.join(" ");  => "1 2 3"
a.join("");  => "123"
var b = new Array(10);  An array of length 10 with no elements
b.join('-')  => '---------': a string of 9 hyphens
```

Array Methods

Array.reverse() method reverses the order of the elements of an array and returns the reversed array. it doesn’t create a new array.

```
var a = [1,2,3];

a.reverse().join()  => "3,2,1" and a is now [3,2,1]
```

Array.filter

The filter() method creates a new array with all elements that pass the test implemented by the provided function.

Syntax :

```
var new_array = arr.filter(callback)
```

Parameters :

Callback : Function is a predicate, to test each element of the array. Return true to keep the element, false otherwise, taking three arguments:

1. Element : The current element being processed in the array.
2. Index : The index of the current element being processed in the array.
3. Array : The array filter was called upon.

Return value :

A new array with the elements that pass the test.

Array.filter

filter() calls a provided callback function once for each element in an array, and constructs a new array of all the values for which callback returns a value that coerces to true. callback is invoked only for indexes of the array which have assigned values; it is not invoked for indexes which have been deleted or which have never been assigned values. Array elements which do not pass the callback test are simply skipped, and are not included in the new array.

Examples : Filtering out all small values

The following example uses filter() to create a filtered array that has all elements with values less than 10 removed.

```
function isBigEnough(value) {
    return value >= 10;
}
var filtered = [12, 5, 8, 130, 44].filter(isBigEnough);
// filtered is [12, 130, 44]
```

Array Methods

Array.sort() sorts the elements of an array in place and returns the sorted array. When sort() is called with no arguments, it sorts the array elements in alphabetical order.

```
var a = new Array("banana", "cherry", "apple");
a.sort();
var s = a.join(", ");  s == "apple, banana, cherry“

var a = [33, 4, 1111, 222];
a.sort();                  Alphabetical order: 1111, 222, 33, 4
a.sort(function(a,b) {     Numerical order: 4, 33, 222, 1111
    return a-b;            Returns <0, 0, or >0, depending on order
});
a.sort(function(a,b) {return b-a});  Reverse numerical order
```

Array Methods

```
a = ['ant', 'Bug', 'cat', 'Dog']
a.sort();  case-sensitive sort: ['Bug','Dog','ant',cat']
a.sort(function(s,t) {  Case-insensitive sort
    var a = s.toLowerCase();
    var b = t.toLowerCase();
    if (a < b) return -1;
    if (a > b) return 1;
    return 0;
});  => ['ant','Bug','cat','Dog']
```

Write a JavaScript function to get the first elements of an array.

Passing a parameter 'n' will return the first 'n' elements of the array

```
first = function (array, n) {
    if (n < 0)
        return [];
    return array.slice(0, n);
};
```

Write a JavaScript function to get the last elements of an array.

Passing a parameter 'n' will return the last 'n' elements of the array

```
last = function (array, n) {
    if (n > 0)
        return array.slice(array.length - n);
};
```

```
function equalArrays(a,b)
{
    if (a.length != b.length)
        return false;             Different-size arrays not equal
    for(var i = 0; i < a.length; i++)
    {                             Loop through all elements
        if (a[i] !== b[i])
        return false;             If any differ, arrays not equal
    }
    return true;                  Otherwise they are equal
}
```

JS

Multidimensional Arrays

```
 Create a multidimensional array
var table = new Array(10);  10 rows of the table
for(var i = 0; i < table.length; i++)
    table[i] = new Array(10);  Each row has 10 columns

 Initialize the array
for(var row = 0; row < table.length; row++) {
    for(col = 0; col < table[row].length; col++) {
        table[row][col] = row*col;
    }
}
 Use the multidimensional array to compute 5*7
var product = table[5][7];  35
```

User-Defined Objects

OOP

Object-oriented programming (OOP)
- Creating reusable software objects

Object
- Programming code and data that can be treated as an individual unit or component

Data
- Information contained within variables

Object-oriented principles
- Encapsulation
- Inheritance
- Polymorphism

Instantiating an object

Creating an object from existing class

- Using **new** Operator

```
var today = new Date();
var objname = new Object();
```

To create an array you have two ways

```
var myarr = new Array[3];
//Or//both create array object
var myarr = [1,2,3];
```

From Arrays To Objects

In Arrays :

```
var myarr=[1,2,3];                    //using [] for array
```

In Objects :

```
var myobj={id:10};                    //using {} for object
```

This is an object that has a property called id with value = 10.

```
var a={};                         //a is an empty object
```

Encapsulation

Each object contain Properties & Methods to access these Properties.

```
var Car = { model: 'Toyota',//object has properties
    color: 'Red',
    move: function() {...}
    beep: function(r){ alert(r) }   //and methods
};
```

Accessing Object Members

There are two ways to access object members

- Using [ ] Square brackets.

```
Car['model'];             //get value from property
Car['color'] = "blue";    //set value to property
Car['move']();            //call a function
```

- Using dot operator.

```
Car.color="black";        //like C++
Car.move();
```

Altering Object Members

Because JavaScript code is parsed not compiled you can add or remove properties of the object.

- Add property to the object.

```
Car.motor="1300 cc";    //adding motor to Car
```

- Remove property from the object.

```
delete Car.model;           //deleting model attribute
Car.model;                  //"undefined"
```

typeof Operator

Now we have the object Car but from which class we instantiated this object.

```
typeof Car;         //Object
```

Each object has a property called **Constructor**

```
Car.constructor;    // Object()
```

The object created using **{ }** it’s constructor is **Object ( )**

Classes in JavaScript

- JavaScript doesn't support the notion of *classes as typical OOP languages* do.

- In JavaScript, you **create *functions that can behave*** *just* **like classes** called **Constructor Function.**

- For example, you can call a function, or you can create an instance of that function supplying those parameters.

Constructor Function

```
function Human()
{
    this.name;                      // this refers to the caller object
    this.age;
    this.sayName=function()        // sayName function in human
    {alert(this.name);}
}
```

To take an object from Human

```
var obj = new Human();
obj.name="Mohamed";
obj.age=22;
obj.sayName();                      // alert Mohamed
```

Global Object

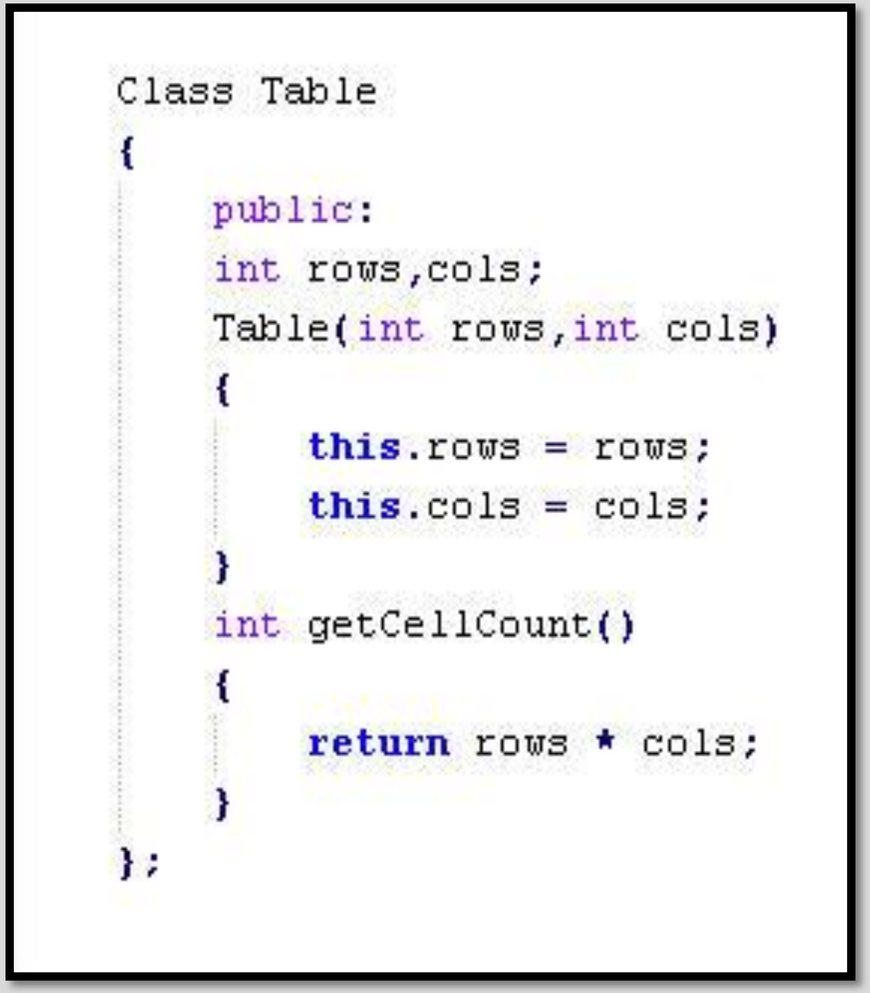
If we called Human function without new operator the **this** refers to the global object **Window**

When you say this.age you add age property to the window object.

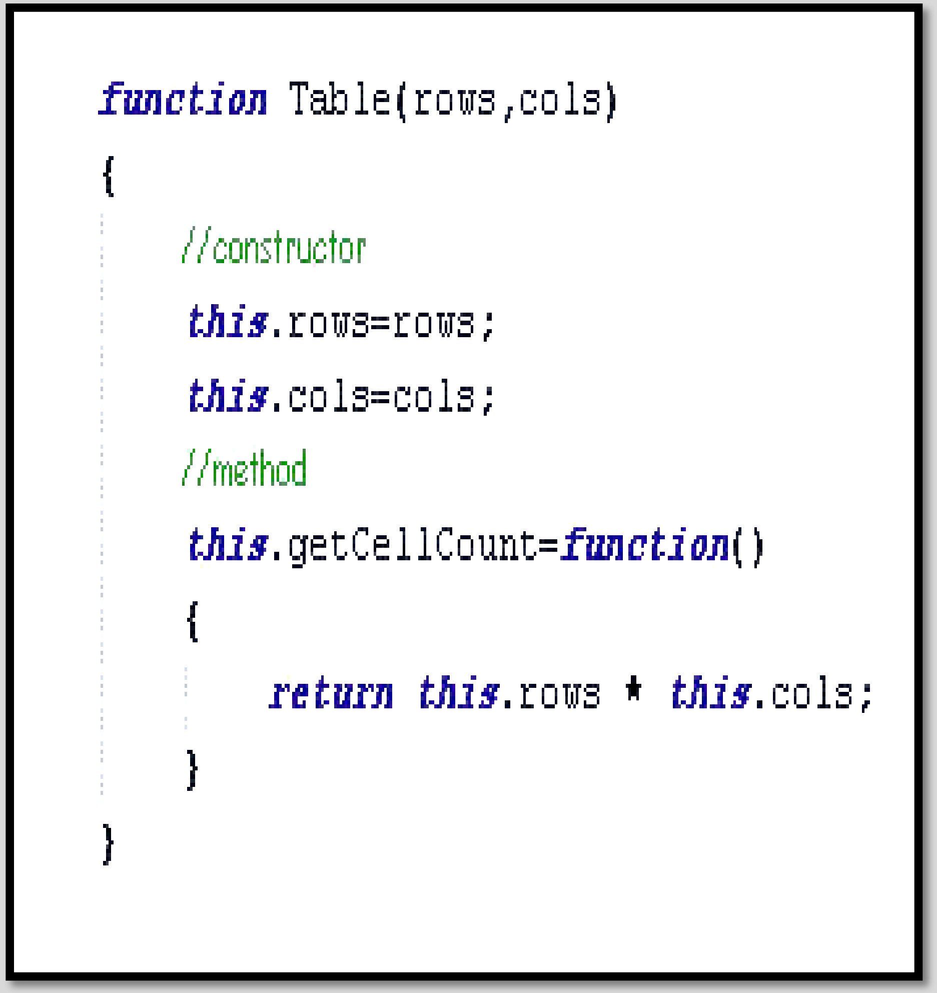
```
Human();

Window.age=30; //window has no age property
```

C++ & JavaScript Classes

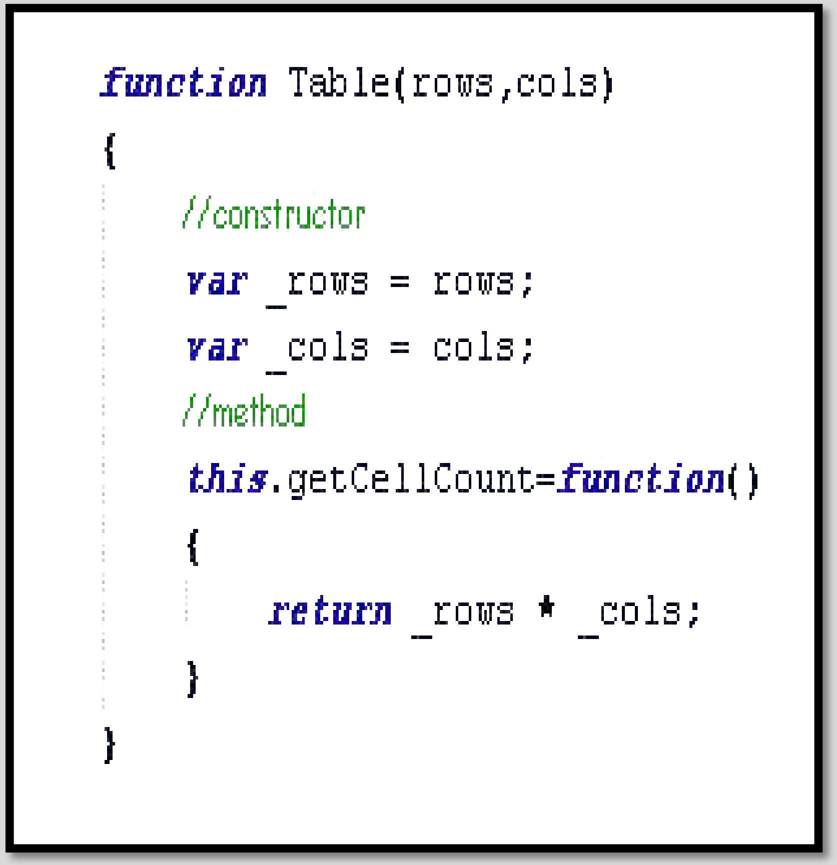


```
Class Table
{
    public:
    int rows,cols;
    Table(int rows,int cols)
    {
        this.rows = rows;
        this.cols = cols;
    }
    int getCellCount()
    {
        return rows * cols;
    }
};
```



```
function Table(rows,cols)
{
    //constructor
    this.rows=rows;
    this.cols=cols;
    //method
    this.getCellCount=function()
    {
        return this.rows * this.cols;
    }
}
```

Private Members



```
function Table(rows,cols)
{
    //constructor
    var _rows = rows;
    var _cols = cols;
    //method
    this.getCellCount=function()
    {
        return _rows * _cols;
    }
}
```

```
//Now if you tries to access
var myTable=newTable(3,2);

myTable.getCellCount();
//it works and returns 6

mytable._rows;
//undefined
```

Object Literals

```
var empty = {};  An object with no properties
var point = { x:0, y:0 };  Two properties
var point2 = { x:point.x, y:point.y+1 };  More complex values
var book = {
    "main title": "JavaScript",  Property names include spaces,
    'sub-title': "The Definitive Guide",  and hyphens, so use string literals
    "for": "all audiences",  for is a reserved word, so quote
    author: {  The value of this property is
        firstname: "David",  itself an object. Note that
        surname: "Flanagan"  these property names are unquoted.
    }
};
```

Thank You