Java Script - Basics

Sridhar Alagar

Introduction

JavaScript was initially created to “make web pages alive”

- Script is part of html and run when the web page is loaded
- Script is written and executed as text. No compilation.

Nothing to do with Java

- Originally called as *LiveScript*

Today JS runs not only in browsers, but also in servers

- Can run on any device that has JS engine

3 great things about JavaScript

Full integration with HTML/CSS

Simple things are done simply

Supported by all major browsers and enabled by default

JS doesn’t suit everyone’s need.

New languages have appeared

- CoffeeScript, TypeScript, Kotlin, etc.
- Transpiled to JavaSript

Almost ‘C’ like syntax

```
i = 3;

i = i * 10 + 3 + (i / 10);

while (i >= 0) {
    sum += i*i;    // Comment
    i--;
}

for (i = 0; i < 10; i++)  {

}
/* this is a comment */
```

```
if (i < 3) {
    i = foobar(i);
} else {
    i = i * .02;
}
```

Most C operators work:

```
* / % + - ! >= <= > < && || ?:
```

```
function foobar(i) { return i;}

continue/break/return
```

JavaScript has dynamic typing

```
let x;
typeof x == ‘undefined’ // undefined is a type
x = 1; // typeof x == type of 1 which is number
x = ‘foobar’; // typeof x is string
x = true; // typeof x is boolean
x = 3.14; // typeof x is number
```

Variables have the type of last value assigned to them

Primitive types: undefined, number, string, boolean, function, object

Variable definition

```
let x; // type undefined
let y = 2;
z = 3; // not allowed if ‘uses strict’

const c = ‘hello’;
c = ‘world’ // error

let userName; // camel case preferred

var i = 10; // almost same as let, but old style
```

number type

```
let n = 3;
n = 3.14;
```

Number type represents both integer and floating point

```
Infinity, -Infinity, NaN are special numbers
```

$(2^{53} - 1)$ is MAX INT

$-(2^{53} - 1)$ is MIN INT

```
Use BigInt to store integers of arbitrary length
```

string type

```
let str = "Hello";
let str2 = 'Single quotes are ok too’;
```

Single quote and double quote are simple quotes

```
let phrase = `can embed another ${expr}`;
```

Can embed expression inside back quote (template literal). The value of the expression is inserted into the string

Lots of useful methods: indexOf(), search(), replace(), toUppercase(), slice(), substr(), etc.

There is no character type

boolean type

`true or false`

JavaScript classifies all values as either truthy or falsy

Used when a value is converted to boolean

Falsy:

`false, 0, NaN, undefined, and null`

Truthy:

Not falsy (all objects, functions, non-empty string, non-zero numbers)

null and undefined

null: a special value to represent “nothing”, “empty”, or “unknown”

typeof null is object;

```
let age = null;
```

undefined: means value not assigned

special type - typeof undefined is undefined

```
let age; // typeof age undefined
age = 100;
age = undefined; // not recommended
alert(age); // "undefined"
null == undefined //true
null === undefined // false
```

Strict equality

Type conversion

String conversion:

performed with String(value)

output are converted to a string

Numeric conversion:

performed with Number(value)

occurs in math operations

Boolean conversion:

performed with Boolean(value)

occurs in logical operations

Nullish coalescing operator '??'

```
let result = a ?? b;
```

is same as

```
result = (a !== null && a !== undefined) ? a : b;

?? and || are different
let height = 0;
alert(height || 100); // 100
alert(height ?? 100); // 0
```

function type

```
function showMessage(from, text = "no text given"){
  alert( from + ": " + text );
}
showMessage("Ann"); // Ann: no text given
```

Can be called with different number of arguments than parameters

Unspecified arguments values are undefined

Array arguments variable – arguments[0] is the first argument

Functions return a value (default value is undefined)

Function definitions are *hoisted.* Can be invoked before definition.

function is a value

```
let sayHi = function() {
  alert( "Hello" );
};
```

Function is assigned to a variable

It is a function expression

Function has no name (*anonymous*)

```
func = sayHi;
alert(sayHi ); //prints the code of the function
func();
sayHi();
```

Callback function

```
function ask(question, yes, no) {
  if (confirm(question)) yes()
  else no();
}
function showOk() {
  alert( "You agreed." );
}
function showCancel() {
  alert( "You canceled the execution." );
}

//functions showOk, showCancel are passed as arguments
ask("Do you agree?", showOk, showCancel);
```

Callback function – uses function expression

```
function ask(question, yes, no) {
  if (confirm(question)) yes()
  else no();
}
ask(
  "Do you agree?",
  function() { alert("You agreed."); },
  function() {alert("You canceled the execution.");}
);
```

function declaration vs function expression

Declared functions are created at the beginning and it is global to the entire block

Function expression are created when the execution reaches the expression

Function declaration is preferrable in most situations

- available before declaration
- flexibility and usually more readable

Use function expression when function declaration is not fit for the task

Arrow function – simple and concise syntax

```
let func = function(arg1, arg2, ..., argN) {
  return expression;
};
```

a shorter version using arrow function

```
let func = (arg1, arg2, ..., argN) => expression;



let sum = (a, b) => a + b;
alert( sum(1, 2) ); // 3

let double = n => n * 2; // parenthesis ignored
let sayHi = () => alert("Hello!"); // no arguments
```

Multi-line Arrow function

```
let sum = (a, b) => {
// the curly brace opens a multiline function
  let result = a + b;
  return result;
// if we use curly braces, then we need an explicit
"return"
};

alert( sum(1, 2) ); // 3
```

Quiz – replace with arrow functions

```
function ask(question, yes, no) {
  if (confirm(question)) yes();
  else no();
}
ask(
  "Do you agree?",
  function() { alert("You agreed."); },
  function() {alert("You canceled the execution.");}
);

ask(
  "Do you agree?",
  () => alert("You agreed."),
  () => alert("You canceled the execution.")
);
```

Debugger

Learn to use browser debugger

Worth the time invested

Will save tons of frustrating time

https://mozilladevelopers.github.io/playground/debugger

https://javascript.info/debugging-chrome

Closure and Scope

```
function makeCounter(){
    let count = 0;
    function increment () { return count++;  }
    return increment; //returns a function
}
let counter = makeCounter();
console.log(counter());
```

console.log(counter)

Closure:

- Is a function that remembers its outer variables
- It remembers the scope (lexical env) in which it was created, so that later they can access them

In JS, all functions are closures

Object type – non-primitive

An object is a collection of *key-value* pairs called properties

```
let foo = {}; // “object literal” syntax
let user = {
    name: “Maya”,
    age: 25,
    “likes birds”: true, // key can be any string
};
user.name //access property value
user[“age”] // square bracket notation
user[“likes birds”] //dot notation won’t work
key = “likes birds”; user[key]
```

Properties can be added, deleted

```
user.id = 2345; //added new property id:2345
delete user.age //age property removed

user.age // undefined
user["age"] === undefined // true
"age" in user // false
```

No restrictions on the name of a property

for, let, return, 5, etc. are allowed

[ ] can be used to add computed properties

```
let key = prompt("Enter property name");
user[key] = value
```

Properties can be enumerated, iterated

```
let user = {
     name: "Maya",
     age: 25,
     id: 2345,
}
Object.keys(user) // enumerate keys
//returns Array ["name", "age", "id"]

for (let key in user){
     alert(key); //outputs "name", "age", "id"
     alert(user[key]);// outputs "Maya", 25, 2345
}
```

Quiz

Write function `isEmpty(obj)` which returns true if the object has no properties, false otherwise.

Should work like this:

```
let schedule = {};
alert( isEmpty(schedule) ); // true
schedule["8:30"] = "get up";
alert( isEmpty(schedule) ); // false
```

this

this is the current object

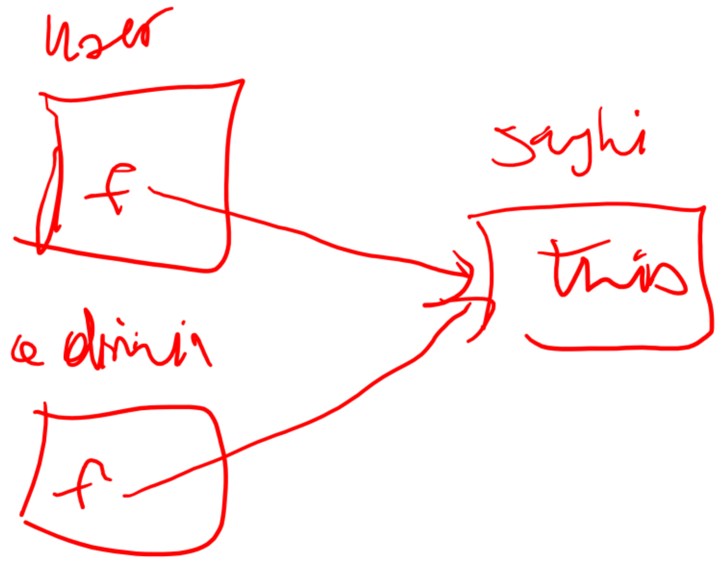
```
let user = {
  name: "John",
  age: 30,

  sayHi() {
    // "this" is the "current object"
    alert(this.name);
  },
};

user.sayHi(); // John
```

Note:
Methods are properties of type function
They can be added later.

This – evaluated at runtime

this is not bound. Evaluated to the object it is in at the runtime

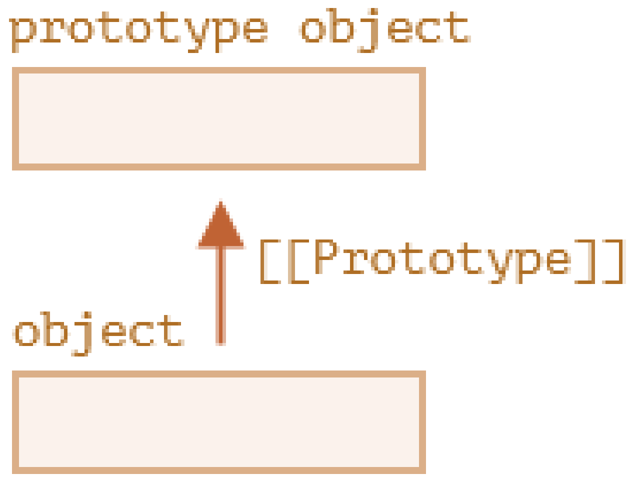
```
let user = { name: "John" };
let admin = { name: "Admin" };
function sayHi() {  alert( this.name );}

// use the same function in two objects
user.f = sayHi;
admin.f = sayHi;

// these calls have different this
// "this" inside the function is the object "before the dot"
user.f(); // John  (this == user)
admin.f(); // Admin  (this == admin)

admin['f'](); // Admin (dot or square brackets access the method)
```



Prototypal Inheritance

[[prototype]] is a special hidden property of all objects

It is either null or refer to another object – prototype object



[[prototype]] can be set using _proto_

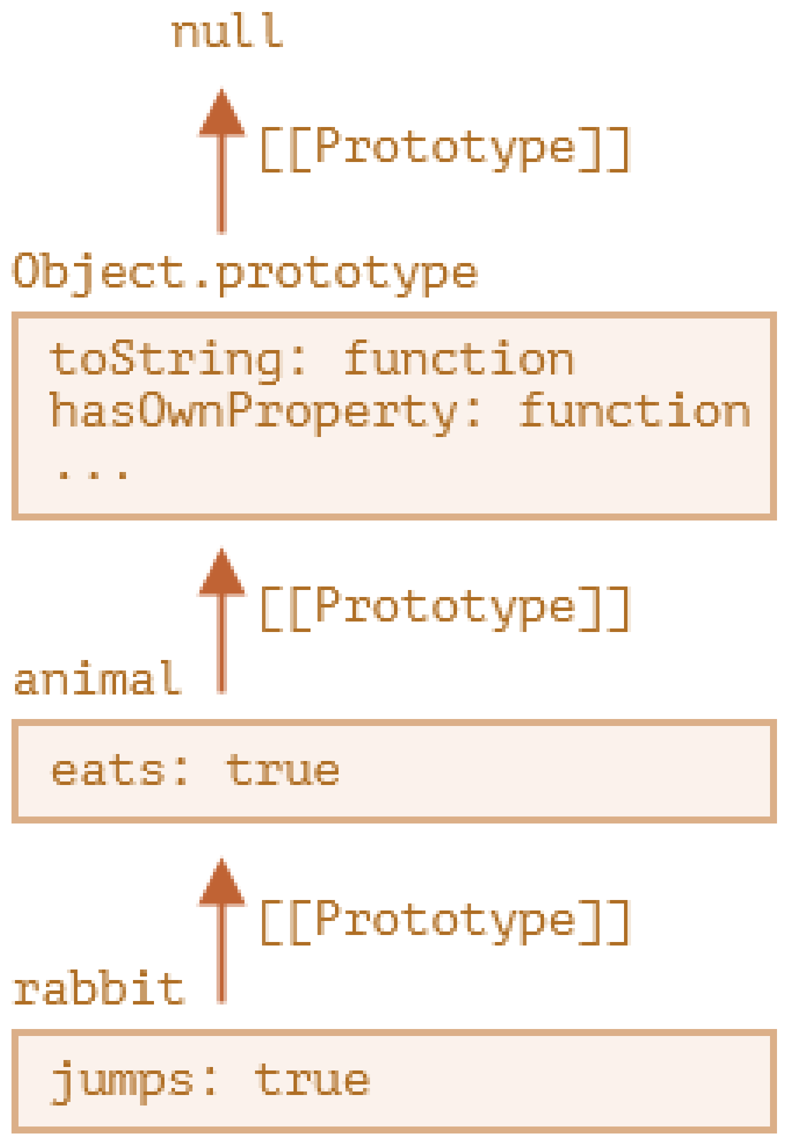
```
let animal = {
  eats: true
};
let rabbit = {
  jumps: true
};

rabbit._proto_ = animal;  // rabbit inherits from animal

// we can find both properties in rabbit now:
alert( rabbit.eats ); // true
alert( rabbit.jumps ); // true
```

Note:
__proto__ is a historical getter/setter for [[Prototype]]

Writing/deleting doesn’t use prototype

```
let animal = {
  eats: true
};
let rabbit = {
  jumps: true
};
rabbit.__proto__ = animal;

rabbit.eats = false; //writes in rabbit
alert( rabbit.eats ); //false
alert( animal.jumps ); // true
```

Object is root



Arrays – object type

```
let fruits = ["Apple", "Orange", "Plum"];

Array can contain mixed values including undefined
arr = [ 'Apple', { name: 'John' }, ,
          function() { alert('hello'); }
        ]; // mixed of values
arr[1].name; // john
arr[2]; // undefined
arr[3](); // 'hello'
arr.length?
fruits[3] = "Pear" // add new value to array
```

Array methods

```
for (let item of fruits){…} // iterate through array
shift(), unshift() // add/del at the front
pop(), push() // add/del at end


splice() // insert, remove, replace

arr.forEach(function(item, index, array) {…});
// function is invoked for each item

result = arr.filter(function(item, index, array) {…});
  // if function returns true item is pushed to results
  // and the iteration continues
  // returns empty array if nothing found
```

Transform an Array

```
result = arr.map(function(item, index, array) {
    // return the new value instead of item
});
```

```
sort(), reverse(), find(), etc.,
slice(), split(), join()
```

Quiz

Write function `isEmpty(obj)` which returns true if the object has no properties, false otherwise.

Destructuring

```
let arr = ["John", "Smith"]

// destructuring assignment
[firstName, lastName] = arr;
firstName; // John
lastName;  // Smith
[n1, , n3] = [1, 2, 3, 4]

[x, y, z] = ‘abc’; // works for any iterable obj
```

Date

```
let date = new Date();
```

Time set to number of milli seconds since midnight Jan 1, 1970 UTC

```
date.getTime() // time in milliseconds
```

Many useful get and set methods

```
getDay(), getMonth(), getHours(), etc.
setDate(), setTime(), etc.
```

Sources

1. https://javascript.info/
2. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference
3. Stanford CS 142 Web Application – Lectures