Joaquín Herrera Ramos
https://github.com/97joaquinhr/

Javascript: Understanding the Weird Parts Notes

March 21st, 2020

Syntax parser: a program that reads your code and determines what does and if its grammar is valid.
Lexical environment: where something sits physically in the code you write
Execution context: a wrapper to help manage the code that is running
Name/value pair: a name which maps to a unique value
Object: is a collection of name/value pairs

Global object “window” (browser) = this is created by default.
Variables are accessible from inspect tool, console, window.variable, or just variable.

Execution Context is created in two phases:

1. Global object, ‘this’, outer environment
2. Setup memory space for variables and functions “Hoisting”

For variables, it creates the space on memory but sets it as undefined.

undefined != not defined
undefined is a keyword, it is a special object that is used for an object that has not been set

Javascript is:
Single threaded: one command at a time
Synchronous: one at a time, in order

March 22nd, 2020

Invocation: running a function
variable environment: where the variable lives
scope: where a variable is available in your code
let is black scoping. let does not create a variable like var, it does not set it as undefined.
asynchronous: more than one at a time
Dynamic typing: you don’t tell the engine what type of data a variable holds, it figures it out during runtime
primitive type: a type of data that represents a single value

1. undefined: lack of existence
2. null: lack of existence (usable)
3. boolean: true or false
4. number: floating point number, only one ‘number’ type
5. string: a sequence of characters, both ‘’ and “” can be used
6. symbol: used in ES6

Operators: a special function that is syntactically different. Operators are functions.
Operator precedence: which operator function gets called first
Associativity: what order operator functions get called in, left-to-right or right-to-left

Joaquín Herrera Ramos
https://github.com/97joaquinhr/

Coercion: converting a value from one type to another, if you add a string and a number, the number is coerced into a string
Coercion can be confusing because you don't normally get what you expect when javascript coerces a value. That is why we use ===, strict inequality, where types matter.
|| returns the first value from left-to-right that can be coerced to true

March 24th, 2020
var obj = {}; === var obj = new Object();
namespace: a container for variables and functions
You can fake a namespace by using objects.
object literals != JSON
JSONs have strings as keys.
First class functions: everything you can do with other types you can do with functions.
Functions are objects
Expression: a unit of code that results in a value

April 6th, 2020
Mutate: to change something
Immutable: can't be changed
'this' points to global variable when a function is created
'this' also changes a variable from the object that is being called

```
var c = {
        name: 'The c object',
        log: function(){
                var self = this; //good practice
                self.name = 'Updated c object';
                console.log(self);

                var setname = function(newname){
                        self.name = newname;
                }
        }
}
c.log();
```

Arrays can hold functions
Arguments: the parameters you pass to a function. Also, a keyword in javascript that contains an arraylike containing all the parameters.
arguments.length: how many arguments were passed

```
function a(x, y, z, ...others){ // if the functions receives more than 3 parameters, they will be
// contained in an array called 'others'
```

```
}
```

Always write the semicolons
Put curly braces on the same line

Whitespace: invisible characters that create literal ‘space’ in your written code
The syntax parser ignores whitespace and comments

April 7th, 2020
Immediately invoked functions expressions (IIFEs):

```
var greeting = function(name){
        return ‘Hello’ + name;
}(‘John’);
console.log(greeting); // prints ‘Hello John’
```

Closure is when a function still has access to a previous execution context.

Callback function: a function you give to another function, to be ran when the other function finishes

April 13th, 2020

Functions have call, bind and apply
Bind creates a copy of whatever function you are calling it from and binds the argument received as “this”
call() receives what will be turned into “this” and the normal arguments separated by commas
apply() receives what will be turned into “this” and the normal arguments inside an array

```
function multiply(a,b){
        return a*b;
}
var multiplyByTwo = multiply.bind(this,2);
console.log(multiplyByTwo(6)); // prints 12
```

function currying: creating a copy of a function but with some preset parameters

underscorejs.org
You can learn from that source code
lodash.com too

April 14th, 2020

Inheritance: one object gets access to the properties and methods of another object

Joaquín Herrera Ramos
https://github.com/97joaquinhr/

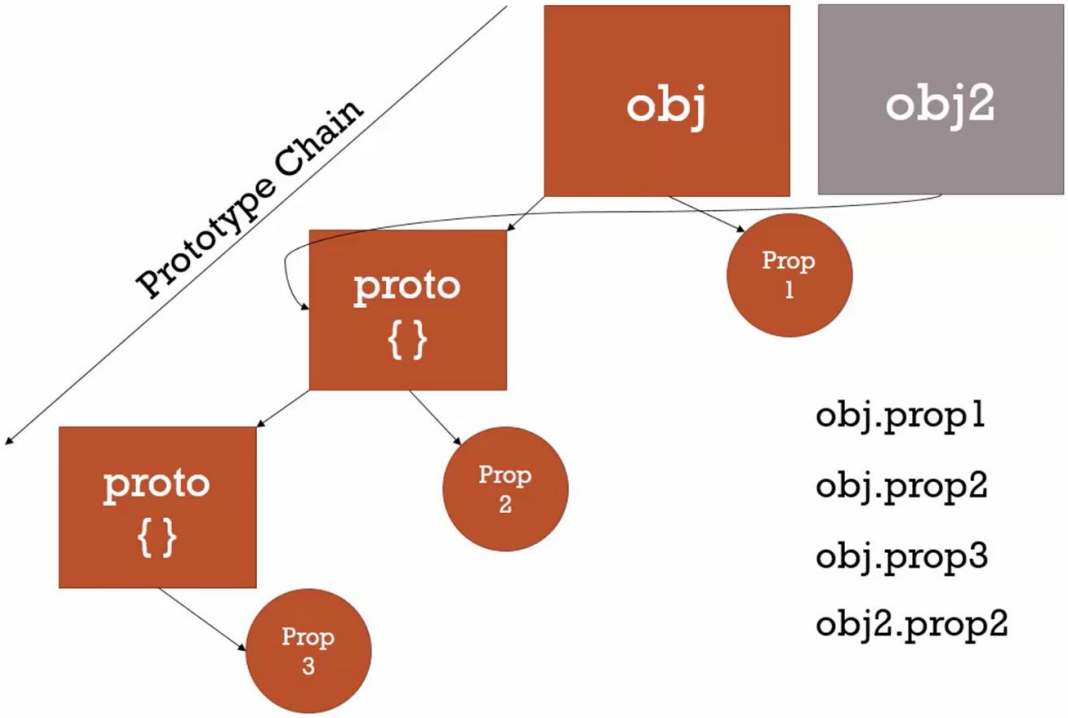
Classic inheritance:

- Verbose
  - friend
  - protected
  - private
  - interface

Prototypal inheritance:

- Simple
  - Flexible
  - extensible
  - easy to understand

All objects have a reference to “proto {}”, which has then has a reference to its own prototype.
Prototype chain, the bottom is an object {}



call, bind and apply are from the prototype
an array’s prototype contains every function like push, pop, reverse, etc.
Reflection: an object can look at itself, listing and changing its properties and methods

New creates an empty object
function constructor: a normal function that is used to construct objects. The ‘this’ variable points a new empty object, and that object is returned from the function automatically

attributes are set on the constructor
methods are added to the prototype:

Joaquín Herrera Ramos
https://github.com/97joaquinhr/

```
function Person(firstname, lastname){
	this.firstname = firstname;
	this.lastname = lastname;
}

Person.prototype.getFullName = function(){
	return this.firstname + ‘ ‘ + this.lastname;
}
```

It is more effective to store methods on the prototype, it saves memory space by using inheritance.

For constructors, use capital letters, to differentiate to other normal functions.
objects are different from primitives because their prototypes contain useful functions
Although there are cases where the Javascript engine takes a primitive and wraps it into an object to allow you to use those functions

primitives !== objects

momentjs.com is awesome apparently
prefer plain for instead of for...in

polyfill: code that adds a feature which the engine may lack

```
Object.create(obj)
//how it works
//polyfill
if(!Object.create){
	Object.create = function(o){
		if(arguments.length > 1){
			throw new Error(‘Object.create implementation’
			+ ‘ only accepts the first parameter.’);
		}
		function F() {}
		F.prototype = o;
		return new F();
	};
}
```

syntactic sugar: a different way to type something that doesn’t change how it work under the hood

Joaquín Herrera Ramos
https://github.com/97joaquinhr/

April 19th, 2020

```
var d = [];
console.log(typeof d); //prints object, which is not useful
console.log(Object.prototype.toString.call(d)); //prints [object Array], which is better
```

“use strict”; // at the top of the file or top of a function
Enforces stricter rules, like making it necessary to declare a variable before initializing it
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode

April 20th, 2020
Method chaining: calling one method after another, and each method affects the parent object.
So obj.method1().method2() where both methods end up with a ‘this’ variable pointing at ‘obj’

April 21st, 2020
Section 9 is very useful for applying everything learned in section 8.

Transpile: convert the syntax of one programming language, to another
In this case, languages that don’t really ever run anywhere, but instead are processed by ‘transpilers’ that generate Javascript.

ES6 https://github.com/lukehoban/es6features