JS Weird Parts 53 - 81

Udemy:
https://www.udemy.com/understand-javascript/learn/v4/overview

Repository:
C:\Users\Lenovo\Desktop\Isti\Programozás-\Kurzusok\Javascript\Udemy-Understanding-weird-Parts

Github:
Useful Link:
http://underscorejs.org/
https://lodash.com/
www.momentjs.com

https://github.com/lukehoban/es6features

Section 5 - Object-Oriented Javascript and Prototypal Inheritance
53 Classical default Prototypal inheritance
54 Understanding the prototype
55 Everything is a primitve or an Object in js
Section 6 - Building Objects
56. Reflection and extend
57. Keyword "new"
58 .Prototype
59. Dangeroues aside "new" and functions
60. Built-in Function Constructors
61 Dangeroues built-in function constructors
62 Dangerous Array for..in
63 Object. Create / Pure Prototypal inheritance
64 ES6 Classes
Section 7 Odds and Ends
66 TypeOf,InstanceOF
67 Strict Mode
Section 8 - Examining Famous Frameworks and Libraries
70. JQuery, Structure, examination
Section 9 - Build a Framework
70. JQuery, Structure, examination
74 Structure safe code
75 Our Object and it's prototype
77. Adding Jquery support
78 Good Commenting
79 Start with ;
Section 10 - Transpilers
81. Transpiler

53 Classical default Prototypal inheritance

Inheritance: on object gets access to the properties and methods of an other object

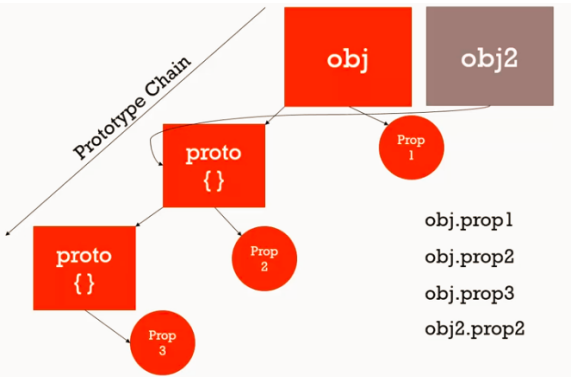
Classical inheritance: (Java, C#)
Verbose - too much confuson, can be complicated

Prototypal inheritance:
flexible

54 Understanding the prototype

Every object, has a property of **_proto,** which could have several other properties.
When an object is looking for a property, **prop2** it will check if the current object has the property or not, then it will go down the **"Prototype Chain"** to find this property somewhere there.

Prototype:



55 Everything is a primitve or an Object in js

BaseObject:
All array, functions, objects have a **"base object"** which is the prototype of the created variable.

56. Reflection and extend

Reflection:
An object can look at itself, listing and changing its properties and methods.

```
var john = {
    firstname: 'John',
    lastname: 'Doe'
}


for (var prop in john) {                    //Loop over every member in the object
```

```
    if (john.hasOwnProperty(prop)) {        //we use the base object's built in
"hasOwnProperty()" which will return a boolean
        console.log(prop + ': ' + john[prop]);
    }
}
```

Underscore.js - extend
Coping over the references of the original objects

```
//Combines the object, with the properties and add them to the john property
_.extend(john, jane, jim);

console.log(john);
```

57. Keyword "new"

new keyword **will create an empty object in the memory**, instead of just calling the function, and returning the created object at the end of the function

```
//new is actually an operator, immediately a new empty object is created.
//since the executional context of the function will generate the keyword of "this"
//when using "new" the "this" variable will point to the newly created empty object in
the memory
// if the function does not return anything, it will return the newly created empty
object

// A function, which has basic constructor
function Person(firstname, lastname) {

    console.log(this);
    this.firstname = firstname;
    this.lastname = lastname;
    console.log('This function is invoked.');
```

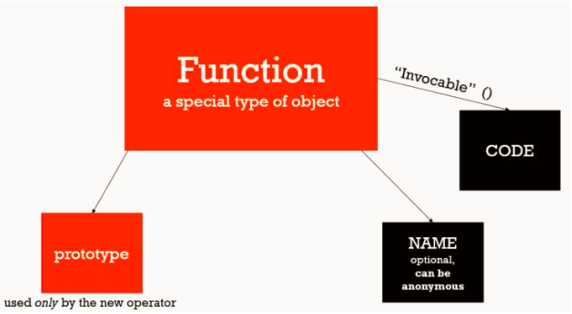
```
}


//It is just creating an object, with calling the functions, with the constructor
var john = new Person('John', 'Doe');
console.log(john);


var jane = new Person('Jane', 'Doe');
console.log(jane);
```

58 .Prototype

When you are using the "**new**" keyword, the prototype object is property of the created function used with the "new" keyword.
Then later every object created with this funciton, will have this prototype property.



code:

```
function Person(firstname, lastname) {
    console.log(this);
    this.firstname = firstname;
    this.lastname = lastname;
    console.log('This function is invoked.');
}


Person.prototype.getFormalFullName = function() {
    return this.lastname + ', ' + this.firstname;
}


console.log(john.getFormalFullName());
```

```
var john = new Person('John', 'Doe');

console.log(john);
```

When we put the functions, properties into the prototype objects we save memoryspace, since all of the other objects are just referencing these created object properties.

59. Dangeroues aside "`new`" and functions

If you are trying to create objects without using the **new** keyword the js will just execute the function and since it is not returning anything, the function will return undefined.

```
var john = new Person('John', 'Doe');

console.log(john);

//this will be undefined

var jane= Person('jane', 'Girl');

console.log(jane);
```

Naming Convention
For constructor function we capitalize the **F**irst letter

```
function Person(firstname, lastname) {


    console.log(this);

    this.firstname = firstname;

    this.lastname = lastname;

    console.log('This function is invoked.');

}
```

60. Built-in Function Constructors

JS will wrap the primitives in objects and it will store the primitive value in the box.

There are String,Number,Data objects, with a whole bunch of properties containing the primitive value in a box.

You can add additional features to the Built-in Function constructors

```
//You can use the built in object's and add additional properties, functions to it's
.prototype property
String.prototype.isLengthGreaterThan = function(limit) {
    return this.length > limit;
}

console.log("John".isLengthGreaterThan(3));

Number.prototype.isPositive = function() {
    return this > 0;
}
```

61 Dangeroues built-in function constructors

www.momentjs.com

```
var a = 3
var b = new Number(3)

a == b  --> true
a === b --> false
//They are not equal, since  primitive 3 is not equal to the created number ojbect
```

62 Dangerous Array for..in

Do not use for .. in

Use forEach, or for loop.

Iterating over object keys, array objects, can be dangerous, since new methods can be added to the prototype property of the array.

```
Array.prototype.myCustom ="Cool"


var arr = [''el1,'el2'];


for (var prop in arr){
    console.log(prop + ":  "  + arr[prop])
};


//will log out,
0: 'el1'
1: 'el2'
"Cool"
```

63 Object. Create / Pure Prototypal inheritance

You can create an empty object, with predefined prototype passed into the function
So it can form the base of other objects.

```
var person = {
    firstname: 'Default',
    lastname: 'Default',
    greet: function() {
        return 'Hi ' + this.firstname;
    }
}


//You can create an empty object, with predefined prototype passed into the function
var john = Object.create(person);
john.firstname = 'John';
john.lastname = 'Doe';
```

```
console.log(john);
```

Using polfyfills for backward compatibility

```
// polyfill for backward compatibility
if (!Object.create) {                                   //if the browser does have it or
not
  Object.create = function (o) {
    if (arguments.length > 1) {                         //should have arguments
      throw new Error('Object.create implementation'
      + ' only accepts the first parameter.');
    }
    function F() {}                                     //Creates an empty function
    F.prototype = o;                                    //Set the passed object as a
prototype
    return new F();                                     //It will create an empty object
due to new keyword
  };
}
```

64 ES6 Classes

It is an other way to create objects and create prototypes.

1. Creating an object, with a constructor
2. Sets the prototype of the new class.
So Person is the prototype, InformalPerson is the new object.

1

```
class Person {

    constructor(firstname, lastname) {
        this.firstname = firstname;
        this.lastname = lastname;
    }

    greet() {
        return 'Hi ' + firstname;
    }
}
var john = new Person('John', 'Doe');
```

2

```
class InformalPerson extends Person  {

    constructor(firstname, lastname) {
        super(firstname, lastname);
    }

    greet() {
        return 'Yo ' + firstname;
    }
}
```

66 TypeOf,InstanceOF

TypeOf: Will tell the type of the variable, or the parameter

```
var a = 3;
console.log(typeof a);          //number

var b = "Hello";
console.log(typeof b);          //string

var c = {};
console.log(typeof c);          //object

var d = [];
console.log(typeof d);         // object, isArray()
console.log(Object.prototype.toString.call(d)); // better! --> Array, we are converting
the object to string.

console.log(typeof undefined); // makes sense
console.log(typeof null); // a bug since, like, forever..., will return an object

var z = function() { };
console.log(typeof z);   //
```

Instance of: Will check if an object is an instance of a prototype object

```
function Person(name) {
    this.name = name;
}

var e = new Person('Jane');
console.log(typeof e);
console.log(e instanceof Person);
```

67 Strict Mode

You can let javascript to get more stiff, less felexible,.
For example you have to defined your variables first!

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode

```
function logNewPerson() {
    "use strict";

    var person2;
    persom2 = {};               //Wrong
    console.log(persom2);
}

var person;
persom = {};                    //Ok, since stric is not defined yet
console.log(persom);
logNewPerson();
```

70. JQuery, Structure, examination

0. Introductory comment
1. Immediately invoked function expression
(Wrapping all of the code in a function to create it's own executonal context)

74 Structure safe code

By wrapping everything in an selfinvokig function and passing inside only the necessary values, we can make our code safe

```
// Global  = window
// $ is the jQuery object
(function(global, $) {

}(window, jQuery));
```

75 Our Object and it's prototype

1. Write a function which **returns a new object.**
2. Create a **constructor function**
3. Create the **prototype object**
4. Assign the created prototype object to the created object's prototype with the **"new"** Keyword
5. Add to the **global variable (window)**

```
(function(global, $) {

 //  1. Write a function which returns a new object.

    var Greetr = function(firstName, lastName, language) {

        return new Greetr.init(firstName, lastName, language);

    }

// 3. Create an empty prototype object

    Greetr.prototype = {};

 // 2.  this is our actual constructor function, with constructor default values

    Greetr.init = function(firstName, lastName, language) {

        var self = this;

        self.firstName = firstName || '';

        self.lastName = lastName || '';

        self.language = language || 'en';


    }

// 4. Assign the empty ojbect to the newly created object's prototype

    Greetr.init.prototype = Greetr.prototype;

// 5. Add to the global object Greetr objec or G$

    global.Greetr = global.G$ = Greetr;


}(window, jQuery));
```

And you can call the global object
app.js

```
var g = G$('John', 'Doe');

console.log(g);
```

76 Properties and Chainable methods

6.1 Setup default methods
6.2 Throw --> provide back an error message
6.3 Chainable method means if you **return the whole object at the end of the methods**, you can chain them like this.

```
g.greet().setLang('es').greet(true);
```

77. Adding Jquery support

Basically, we are checking if jquery object has been passed, create a html with internal functions, using the passed jquery object to interact with html, then return the object so it can be chainable

7.0 Check if jquery is added, since we are trying to reference the global jquery object
7.1 reading the passed selector 7.2 Crearing the message with internal functions
7.3 Using the passed jquery object from the closure
7.4 Make it chainable

78 Good Commenting

- Variable explanation
- Function explatation
- Dependency description
- tricky sides

79 Start with ;

Sometimes the other code does not finishes with ; it could generate some errors.

```
;(function(global, $) {

}();
```

81 Transpile

Transpile: Convert the syntax of 1 language to an other
http://www.typescriptlang.org
http://www.typescriptlang.org/Playground