Hi to everyone. My name is Victor. The topic of my presentation is: «Inheritance in JavaScript».

JavaSctript has a prototype inheritance. The prototype is an object from which other objects inherit their properties. It can be used as a backup property storage.

Inheritance is implemented through a hidden property reference ‘\_\_proto\_\_’, which is not used in the code. It can be seen only through the DevTool browser.

How does the inheritance mechanism work?

Each object in JS is a set of pairs ‘key-value’. Each object has a ‘\_\_proto\_\_’ property that refers to another object. If we talk about a simple object, then ‘\_\_proto\_\_’ refers to the Object.prototype. It is quite easy to understand. When we request a property and the interpreter does not find it in the object, it follows the link ‘\_\_proto\_\_’. If it still cannot find it, then the interpreter goes higher along the prototype chain until it reaches Object.prototype. At the end, if it still cannot find it, the interpreter returns ‘undefined’. It is important to remember that the prototype chain works exclusively for reading, it means, when we assign, then a long check does not occur throughout the chain. The property is assigned to the object itself. To better understand, how the prototype chain works, consider the following scheme:

It can be seen, that all arrays have Array.prototype in which all array methods are stored and called when needed. The same works for the functions and numbers, which have Function.prototype and Number.prototype, respectively, with a set of methods. Array.prototype, Number.prototype, and Function.prototype come from Object.prototype, which by the reference ‘\_\_proto\_\_’ refers to null.

Thus(вас), all objects in JavaScript are descendants of Object, and inherit methods and properties from the Object.prototype.

To create an object, which doesn't have a prototype.

What is a prototype inheritance used for? For example, if there is need to create objects with similar properties. No need to create each object separately many times and use megabytes of memory, but to write down all the properties in one object and assign it as a prototype for many descendants. In the following, when referring to any descendant, the interpreter will find the necessary properties in the prototype, if needed. Thus, it is possible to transfer methods to the prototype.

To use all of this, let's look at this in more detail.

A function is also a set of key-value pairs. Functions ‘\_\_proto\_\_’ refers to Function.prototype. The function has a property “prototype”, which refers to an “almost empty” object. In the object, there is a link ‘\_\_proto\_\_’, which refers to the Object.prototype, and a constructor property, which refers to the function itself. BUT this object works only when the function is called through(сру) the operator “new”. When a function is called this way, then there are differences from its standard behavior:

The first. A new object, in which functions are transferred as ‘this’ is created. The function is also called, but ‘this’ is a new object.

The second. This new object is returned implicitly.

The third. The prototype property will be assigned to the function's prototype property. In other words, the link ‘\_\_proto\_\_’ will refer to the object that is in the prototype property

Here, for example, how you can create a function with which you can easily produce similar objects with properties: ‘name’, ‘id’, ‘avatar’.

For example, you need to add a method to remove the avatar.

However, this method will be in each new object formed from ‘User’. Nevertheless, this function is not always necessary for the newly created object. By creating many new objects, this new function will be in each, and will simply take up space in the browser's memory. That is the reason, why prototype inheritance is applied.

For this，you can use the prototype properties. The slide shows that the ‘deleteAvatar’ method is not available for the new objects, but their prototype has this method. When calling the ‘deleteAvatar’ method of all new objects, formed from user, the interpreter will find the method in the prototype.

Also, prototype inheritance allows to create a child object for an already created object and inherit all properties from it. For example, create an Admin object that inherits all properties from ‘User’:

Then to add a method to the new ‘Admin’ object, that ‘Admin’ will have, but ‘User’ will not.

In ES6 everything looks different

In ES6 is more convenient to use prototype inheritance, the code has become more readable, short. If you look at a specific example, instead of a long record: ‘Admin.prototype equally Object.create (User.prototype)’ a shorter and more readable “Admin extends User” is used.

In addition, it is important to mention the operator ‘instanceof’. It allows you to check which class the object belongs to given prototype inheritance. It works as follows: ‘obj instanceof Constructor’ returns ‘true’, if the object belongs to the class ‘Constructor’ or to a class that inherits from it.

Work algorithm

1. Get ‘obj.proto’

2. Compare ‘obj.proto’ with ‘Constructor.prototype’

3. If does not match, then replace ‘obj’ with ‘obj.proto’ and repeat the check in the step 2 until either a match is found (result ‘true’) or the prototype chain does not end (result ‘false’)

The constructor function is not attended in the verification process! For the checking object, only the prototype chain is important.

THAT IS ALL ! THANK YOU!!!