



Design and Implementation of Online Behavioral Experiments



nodeGame.org
Stefano Balietti
MZES and Heidelberg

Advanced JavaScript

@balietti
@nodegameorg
stefanobalietti.com@gmail.com

Variable Declaration in JS

```
var a = 1; // number
var b = 'Hello world!'; // string
var c = function(p) { return p+1; }; // function
var d = { key: 'value' }; // object
var e = [ "value1", 3, c ]; // array
```

- Variables are **loosely typed**
- Variables are scoped within the **function** in which they are declared
- Functions and array are objects
- All the variables declared above are **global**

Variable Declaration in JS

```
var a = 1; // number
var b = 'Hello world!'; // string
var c = function(p) { return p+1.1.1. // function
var d = { key: 'value' }; // object
var e = [ "value1", 3, c ]; // array
```

Note! ES6 (the newest version of JavaScript) introduced some changes here. However, we stick to version ES5 to maximize our experimental audience.

- Variables are **loosely typed**
- Variables are scoped within the **function** in which they are declared
- Functions and array are objects
- All the variables declared above are **global**

Variable Declaration in JS

How to make variables *private* ?

Declare them inside an own function!

This technique is called: *closure*

```
function foo(bar) {  
    var a = bar;  
}  
foo(10);  
console.log(a); // undefined
```

Variable Declaration in JS

```
function foo() {  
    var a = 1;  
}  
foo();  
console.log(a); // undefined
```

What happens if we do not use the `var` keyword?

Variable Declaration in JS

```
function foo() {  
    var a = 1;  
}  
foo();  
console.log(a); // undefined
```

What happens if we do not use the `var` keyword?

JS will try to access the *global variable a*

Variable Declaration in JS

```
function foo() {  
  var a = 1;  
}  
foo();  
console.log(a); // undefined
```

What happens if we do not use the `var` keyword?

JS will try to access the *global variable a*
What if there is no *global variable a*?

Variable Declaration in JS

```
function foo() {  
  var a = 1;  
}
```

```
foo();
```

```
console.log(a); // undefined 1
```

What happens if we do not use the `var` keyword?

JS will try to access the *global variable a*
What if there is no *global variable a*?

Variable *leaking* into the global scope

Avoid Programming Mistakes

There are ways to avoid these types of mistakes in JS

- Use a linting tool: JSHint/JSLint/ESLint/JSCS
<https://www.sitepoint.com/comparison-javascript-linting-tools>
- JS Strict Mode
`"use strict";`

Linting Tool: JSHint


Try it on : <http://jshint.com>

```
1 // Hello.
2 //
3 // This is JSHint, a tool that helps to detect errors and potential
4 // problems in your JavaScript code.
5 //
6 // To start, simply enter some JavaScript anywhere on this page. Your
7 // report will appear on the right side.
8 //
9 // Additionally, you can toggle specific options in the Configure
10 // menu.
11
12 function main() {
13     return 'Hello, World!';
14 }
15
16 main();
```

CONFIGURE
JSHint
Metrics
version 2.9.1-rc1
There is only one
About It takes no argum
Documentation This function con
Install Cyclomatic comp
Contribute
Blog

Strict Mode on

```
function foo(bar) {  
    "use strict";  
    t = bar;  
}  
foo(10);
```

 ▶ **ReferenceError: assignment to undeclared variable** debugger eval code:3:3
t [\[Learn More\]](#)

JS Function Input Parameters

```
// Internally modifies input.  
function doSomething(obj, num, str) {  
    obj.a = 10;  
    num = 1;  
    str = 'a';  
}  
var obj = {}, num = 0, str = '';  
doSomething(obj, num, str);  
  
console.log(obj);  
console.log(num);  
console.log(str);
```



What will the final values of the object, the string, and the number be, after they have been modified by the function?

JS Function Input Parameters

```
// Internally modifies input.
function doSomething(obj, num, str) {
  obj.a = 10;
  num = 1;
  str = 'a';
}
var obj = {}, num = 0, str = '';
doSomething(obj, num, str);

console.log(obj); // { a: 10 }
console.log(num); // 0
console.log(str); // 'a'
```

Objects are passed as a *reference (to an address in memory)*, while numbers and strings are *copies*

Modifying a copy does not affect the value outside the function, modifying the reference does.

Our Previous Example: Arrays and For Loops

```
var message = 'I like ';  
// This is a "for loop".  
for (var i = 0 ; i < fruits.length ; i++) {  
    message += fruits[i];  
    if (i < (fruits.length - 1 )) {  
        message += ', ';  
    }  
    else {  
        message += '.';  
    }  
}  
alert(message);
```

Our Previous Example: Arrays and For Loops

```
var message = 'I like ';  
// This is a "for loop".  
for (var i = 0 ; i < fruits.length ; i++) {  
    message += fruits[i];  
    if (i < (fruits.length - 1 )) {  
        message += ', ';  
    }  
    else {  
        message += '.' ;  
    }  
}  
alert (message) ;
```



That's a lot of code inside the for-loop. How to make it more compact?

Functions with Returns

```
function join(word, index, arraySize, endSign) {  
    if (undefined === endSign) endSign = '.';  
    if (index === arraySize - 1) word += ',';  
    else word += endSign;  
    return word;  
}
```

If-else branches can be written without parentheses, and they apply to the next line, as delimited by semicolon (;).

```
var message = 'I like '  
// This is a "for loop".  
for (var i = 0 ; i < fruits.length ; i++) {  
    message += join(fruits[i], i, fruits.length, "!");  
}
```


Functions with Returns

```
function join(word, index, arraySize, endSign) {  
    if (undefined === endSign) endSign = '.';  
    if (index === arraySize - 1) word += ',';  
    else word += endSign;  
    return word;  
}
```

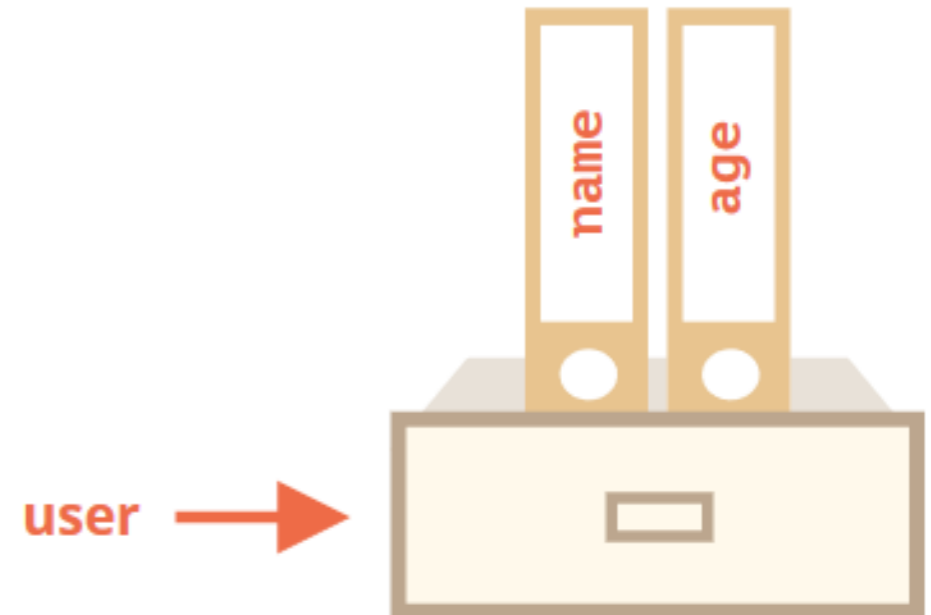
The return keyword makes available outside of the function the modified variable `word`.

```
var message = 'I like '  
// This is a "for loop".  
for (var i = 0 ; i < fruits.length ; i++) {  
    message += join(fruits[i], i, fruits.length, "!");  
}
```

Functions are Objects

<http://javascript.info/>

```
// Our previous example of user John.  
var user = {  
  name: "John", // by key "name" store value "John"  
  age: 30       // by key "age" store value 30  
};
```



Functions are Objects

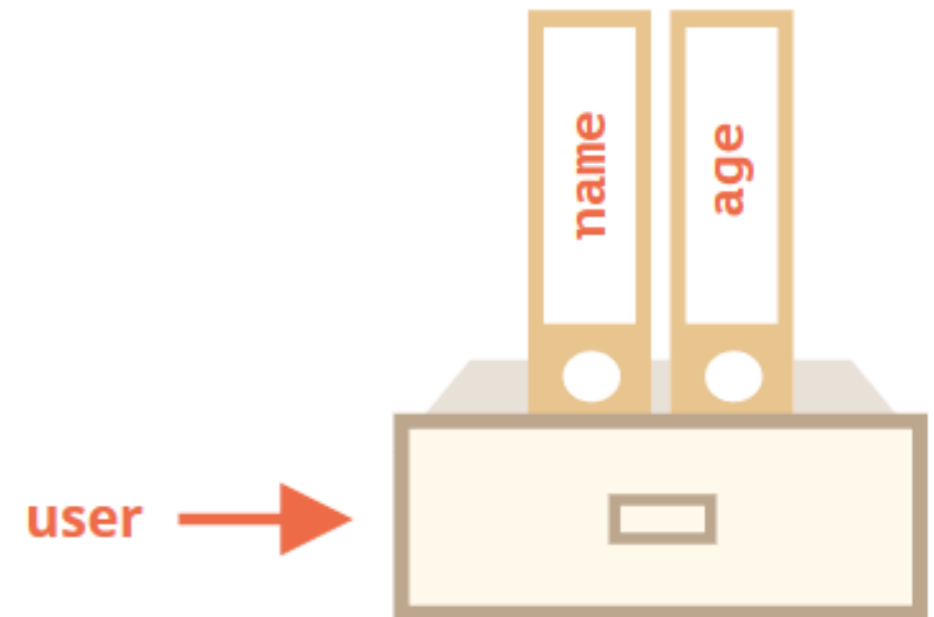
<http://javascript.info/>

```
// Our previous example of user John.  
var user = {  
  name: "John", // by key "name" store value "John"  
  age: 30       // by key "age" store value 30  
};
```

```
// It is equivalent to:
```

```
function User() {  
  this.name = "John";  
  this.age = 30;  
}
```

```
var user = new User();
```



Creating Objects with a Constructor

<http://javascript.info/>

```
// Our previous example of user John.  
var user = {  
  name: "John", // by key "name" store value "John"  
  age: 30       // by key "age" store value 30  
};  
// It is equivalent to:  
function User() {  
  this.name = "John";  
  this.age = 30;  
}  
var user = new User();
```

In the technical language the function User is called the "constructor" or "class," and the variable user is the live "instance" of the class User.

Creating Objects with a Constructor

```
// Our previous example of user John.  
var user = {  
  name: "John", // by key "name" store value "John"  
  age: 30       // by key "age" store value 30  
};  
// It is equivalent to:  
function User() {  
  this.name = "John";  
  this.age = 30;  
}  
var user = new User();
```

Behind the scenes, "new" is adding
the lines in bold to the User function

```
function User() {  
  var this = {};  
  this.name = "John";  
  this.age = 30;  
  return this;  
}
```

Creating Objects with a Constructor

```
// Our previous example of user John.
var user = {
  name: "John", // by key "name" store value "John"
  age: 30       // by key "age" store value 30
};
// It is equivalent to:
function User() {
  this.name = "John";
  this.age = 30;
}
var user = new User();
```



What is the advantage of using a constructor function?

Constructors Take Input Parameters

```
function User(name, age) {  
    this.name = name;  
    this.age = age;  
}  
var user = new User("John", 30);
```

Constructors Can Define Methods

```
function User(name, age) {  
    this.name = name;  
    this.age = age;  
  
    this.sayHello = function(to) {  
        var hello = 'Hello ' + to;  
        hello += '. I am ' + this.name;  
        hello += ', and I am ' + this.age;  
        alert(hello);  
    };  
}
```

The order in which properties and methods inside of the constructor are defined does not matter. They are evaluated only after the constructor is *instantiated* into a live object.

JS Fiddle

The screenshot shows the JS Fiddle website interface. At the top, there is a browser address bar with the URL `https://jsfiddle.net`. Below the browser, the JS Fiddle logo is on the left, followed by navigation buttons: Run, Save, Tidy, and Collaborate. A green 'Update' button is next to the text 'New and updated JS/CSS linters'. On the right, there are links for Settings and Sign in.

The main content area is divided into three columns:

- Fiddle meta:** Contains a text input for the title (currently 'Untitled fiddle'), a text input for the description (currently 'No description'), and a note: 'Add title to make the fiddle visible on your profile page'. Below this are 'Resources' with buttons for 'URL' and 'cdnjs', and sections for 'Async requests' and 'Other (links, license)'.
- HTML:** A dropdown menu is set to 'HTML', and the content area contains a single line of text: '1'.
- CSS:** A dropdown menu is set to 'CSS', and the content area contains a single line of text: '1'.

Below the HTML and CSS panels, there is a section for the JavaScript engine, with a dropdown menu set to 'JavaScript + No-Library (pure JS)' and the content area containing a single line of text: '1'. To the right of this section is a 'Result' panel.

At the bottom of the page, there is an advertisement for Transifex, featuring a colorful geometric logo and the text: 'Learn why developers love using Transifex to localize their digital content.' Below the ad, it says 'ads via Carbon'.

JS Fiddle

The image shows a screenshot of the JS Fiddle website interface. The browser's address bar displays `https://jsfiddle.net`. The main interface is dark-themed and features a top navigation bar with icons for Run, Save, Tidy, Collaborate, Update, and Settings. The main workspace is divided into four panels:

- HTML**: The top-left panel, containing a yellow box with the text "Here HTML Code".
- CSS**: The top-right panel, containing a yellow box with the text "Here CSS".
- JavaScript + No-Library (pure JS)**: The bottom-left panel, containing a yellow box with the text "Here JavaScript".
- Result**: The bottom-right panel, currently empty.

On the left side, there is a sidebar with "Fiddle meta" (Untitled fiddle, No description), "Resources" (URL, cdnjs), "Async requests", and "Other (links, license)". At the bottom left, there is an advertisement for "transifex" with the text "Learn why developers love using Transifex to localize their digital content."

JS Fiddle

The screenshot shows the JS Fiddle website interface. The browser address bar displays `https://jsfiddle.net`. The main navigation bar includes a cloud icon, a **Run** button (circled in orange), a **Save** button, and links for **Tidy**, **Collaborate**, **Update**, **New and updated JS/CSS linters**, **Settings**, and **Sign in**.

The interface is divided into four main panels:

- Fiddle meta**: Contains fields for "Untitled fiddle" and "No description", and a note: "Add title to make the fiddle visible on your profile page". It also has "Resources" (URL, cdnjs) and "Async requests" sections.
- HTML**: A code editor panel with a line number "1" and a yellow box containing the text "Here HTML Code".
- CSS**: A code editor panel with a line number "1" and a yellow box containing the text "Here CSS".
- JavaScript + No-Library (pure JS)**: A code editor panel with a line number "1" and a yellow box containing the text "Here JavaScript".
- Result**: A panel for the execution output, with a yellow box containing the text "Output!".

At the bottom left, there is an advertisement for **transifex** with the text: "Learn why developers love using Transifex to localize their digital content." and "ads via Carbon".

Hands On 4 A

1. Add two inputs (age and name)

hint: HTML tag `<input type="text" />`; check the "placeholder" attribute)

2. Add one button

(hint: HTML tag `<button>Text</button>`)

3. When the button is clicked a new object of type User is created with values from the two inputs.

(hints: get the inputs with `document.getElementById` and look into the property named `value`; remember the `onclick` function?)

4. Display an alert message of the successful creation

Hands On 4 B

5. Add another input for *your* name and another button
6. When the button is clicked,
 - A. The method `sayHello` of the object of class `User` is invoked with the value of the input as parameter.
 - B. The return value of `sayHello` is added to the page.

Hints: `document.createElement`; `document.body.appendChild`; `innerHTML` property

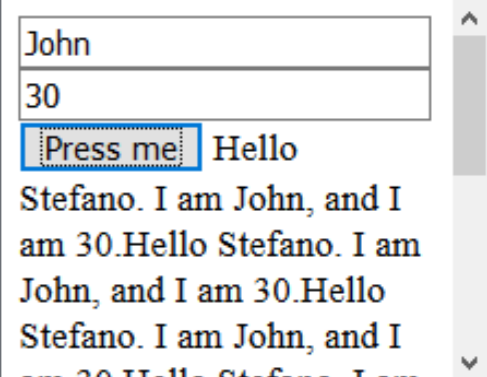
Solution

Yes, I clicked the "Press Me" button a few times.

```
HTML ▼
1 <input type="text" id="myname" placeholder="Name"/>
2 <input type="text" id="myage" placeholder="Age"/>
3 <button id="mybutton">Press me</button>

CSS ▼
1
2

JavaScript + No-Library (pure JS) ▼
1 function User(name, age) {
2   this.name = name;
3   this.age = age;
4   this.sayHello = function(to) {
5     var hello = 'Hello ' + to;
6     hello += '. I am ' + this.name;
7     hello += ', and I am ' + this.age + '.';
8     // alert(hello);
9     return hello;
10  };
11 }
12
13 var name = document.getElementById("myname");
14 var age = document.getElementById("myage");
15 var button = document.getElementById('mybutton');
16
17 button.onclick = function() {
18   // Example of input validation.
19   if (isNaN(parseInt(age.value, 10))) {
20     alert('Hey this is Not A Number (NaN)!')
21     // Return to avoid execution of the code below.
22     return;
23   }
24   // Assume inputs are valid.
25   var user = new User(name.value, age.value);
26   var hello = user.sayHello("Stefano")
27   var span = document.createElement("span");
28   span.innerHTML = hello;
29   document.body.appendChild(span);
30 }
```



Bonus Hands On 4 C

7. Assign a random color to every new DIV containing the result of sayHello

Hints:

create an array of colors = ["red", "green", ...];

get a random index from 0 to the size of the array - 1;

`Math.floor(Math.random() * myArray.length);`