Client-side JavaScript

Client-side Storage
Scripted Graphics
TypeScript
Frameworks and Development Tools

Tecnologie Internet

a.a. 2022/2023

Client-side Storage

Client-side storage

Web applications can use browser APIs to store data locally on the user's computer.

Web apps can store user preferences, for example, or even store their complete state, so that they can resume exactly where the user left off at the end of his/her last visit.

Pages from one site cannot read the data stored by pages from another site. But two pages from the same site can share storage and can use it as a communication mechanism.

Web applications can choose the lifetime of the data they store: data can be stored temporarily (e.g., until the window closes) or permanently.

Client-side storage

There are a number of forms of client-side storage:

- **Web Storage**: an API which was originally defined as part of HTML5, but was spun off as a standalone specification. https://www.w3schools.com/html/html5_webstorage.asp

- **Cookies**: originally designed for use by server-side scripts; they are scriptable on the client-side, but they are hard to use and are suitable only for storing small amounts of textual data; also, any data stored as cookies is always transmitted to the server with every HTTP request, even if the data is only of interest for the client.

- **File System Access**: for apps which read, write, and create files and/or directories in a virtual, sandboxed file system; not supported by all browsers, yet. https://wicg.github.io/file-system-access/

.. and more (**Offline Web Applications**, **IE User Data**, **Web Databases**).

Client-side storage

Unlike cookies, with **Web Storage** the storage limit is far larger (at least 5MB) and information is never transferred to the server.

HTML local storage provides two objects for storing data on the client:

- `window.localStorage` - stores data with no expiration date
- `window.sessionStorage` - stores data for one session (data is lost when the browser tab is closed)

Before using them, check browser support for localStorage and sessionStorage:

```
if(typeof(Storage) !== "undefined") {
    // Code for localStorage/sessionStorage.
} else {
    // Sorry! No Web Storage support..
}
```

Client-side storage

- Create a localStorage name/value pair with name="lastname" and value="Smith"
- Retrieve the value of "lastname" and insert it into the element with id="result"

```
<!DOCTYPE html>
<html>
<body>

<div id="result"></div>

<script>
// Check browser support
if (typeof(Storage) !== "undefined") {
    // Store
    localStorage.setItem("lastname", "Smith");
    // Retrieve
    document.getElementById("result").innerHTML =
localStorage.getItem("lastname");
} else {
    document.getElementById("result").innerHTML = "Sorry, your
browser does not support Web Storage...";
}
</script>

</body>
</html>
```

Client-side storage

The syntax for removing the "lastname" localStorage item is as follows:

```
localStorage.removeItem("lastname");
```

Note: Name/value pairs are always stored as strings. Remember to convert them to another format when needed!

Client-side storage

The following example counts the number of times a user has clicked a button. In this code the value string is converted to a number to be able to increase the counter.

```
if (localStorage.clickcount) {
    localStorage.clickcount = Number(localStorage.clickcount) + 1;
} else {
    localStorage.clickcount = 1;
}
document.getElementById("result").innerHTML = "You have clicked the
button " + localStorage.clickcount + " time(s).";
```

http://www.w3schools.com/html/tryit.asp?filename=tryhtml5_webstorage_local_clickcount

Client-side storage

The sessionStorage object is equal to the localStorage object, **except** that it stores the data for only one session. The data is deleted when the user closes the specific browser tab.

The following example counts the number of times a user has clicked a button, in the current session:

```
if (sessionStorage.clickcount) {
    sessionStorage.clickcount = Number(sessionStorage.clickcount) +
1;
} else {
    sessionStorage.clickcount = 1;
}
document.getElementById("result").innerHTML = "You have clicked the
button " +
sessionStorage.clickcount + " time(s) in this session.";
```

http://www.w3schools.com/html/tryit.asp?filename=tryhtml5_webstorage_session

Scripted Graphics

Scripted graphics

Generating sophisticated graphics in the browser is important for several reasons:

- the code used to produce graphics on the client is typically much smaller than the images themselves, creating a substantial **bandwidth saving**;

- dynamically generating graphics from real-time data uses a lot of CPU cycles; offloading the task to the client **reduces the load on the server**;

- generating graphics on the client is **consistent with modern web application architectures**, in which servers provide data and clients manage the presentation of that data.

Scripted graphics

The HTML **<canvas>** element is used to draw graphics on the fly, via scripting (usually JavaScript). It became a standard with HTML5.

The `<canvas>` element is only a container for graphics. You must use a script to actually draw the graphics.
Canvas has several methods for drawing paths, boxes, circles, text, and adding images.

The `<canvas>` element must have an `id` attribute so it can be referred to by JavaScript.
The width and height attributes are necessary to define the size of the canvas.

```
<canvas id="myCanvas" width="200" height="100"></canvas>
```

It is possible to have multiple `<canvas>` elements on one HTML page.

Scripted graphics

```
<!DOCTYPE html>
<html>
<body>

<canvas id="myCanvas" width="200" height="100"
style="border:1px solid #c3c3c3;">
Your browser does not support the canvas element.
</canvas>

<script>
let canvas = document.getElementById("myCanvas");
let ctx = canvas.getContext("2d");
ctx.fillStyle = "#FF0000";
ctx.fillRect(0,0,150,75);
</script>

</body>
</html>
```

The upper-left corner of the canvas has the coordinates (0,0).

The `getContext()` is a built-in HTML object, with properties and methods for drawing.

We set the fill style of the drawing object to the color red.

The `fillRect(`*`x,y,width,height`*`)` method draws a rectangle, filled with the fill style.

Scripted graphics

To draw a straight line on a canvas, use the following methods:

- `moveTo(x,y)` - defines the starting point of the line
- `lineTo(x,y)` - defines the ending point of the line

To actually draw the line, you must use one of the "ink" methods, like `stroke()`.

```
let canvas = document.getElementById("myCanvas");
let ctx = canvas.getContext("2d");
ctx.moveTo(0,0);
ctx.lineTo(200,100);
ctx.stroke();
```

Scripted graphics

To draw a circle on a canvas, use the following methods:

- `beginPath()`
- `arc(x,y,r,start,stop)`

```
let canvas = document.getElementById("myCanvas");
let ctx = canvas.getContext("2d");
ctx.beginPath();
ctx.arc(95,50,40,0,2*Math.PI);
ctx.stroke();
```

Scripted graphics

Shapes on the canvas are not limited to solid colors.
Gradients can be used to fill rectangles, circles, lines, text, etc.

There are two different types of gradients:

- `createLinearGradient(x,y,x1,y1)` - creates a linear gradient
- `createRadialGradient(x,y,r,x1,y1,r1)` - creates a radial/ circular gradient

Once we have a gradient object, we must add two or more color stops.

The `addColorStop()` method specifies the color stops, and its position along the gradient. Gradient positions can be anywhere between 0 to 1.

To use the gradient, set the `fillStyle` or `strokeStyle` property to the gradient, then draw the shape (rectangle, text, or a line).

Scripted graphics

```
let c=document.getElementById("myCanvas");
let ctx=c.getContext("2d");

// Create gradient
let grd=ctx.createLinearGradient(0,0,200,0);
grd.addColorStop(0,"red");
grd.addColorStop(1,"white");

// Fill with gradient
ctx.fillStyle = grd;
ctx.fillRect(10,10,150,80);
```

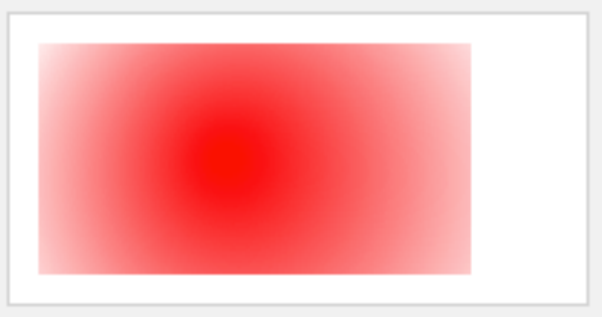
Scripted graphics

```
let c=document.getElementById("myCanvas");
let ctx=c.getContext("2d");

// Create gradient
let grd=ctx.createRadialGradient(75,50,5,90,60,100);
grd.addColorStop(0,"red");
grd.addColorStop(1,"white");

// Fill with gradient
ctx.fillStyle = grd;
ctx.fillRect(10,10,150,80);
```

Note: the 1st circle must be included into the 2nd one.



Scripted graphics

To draw text on a canvas, the most important property and methods are:

- `font` - defines the font properties for the text
- `fillText(`*`text,x,y`*`)` - draws "filled" text on the canvas
- `strokeText(`*`text,x,y`*`)` - draws text on the canvas (no fill)

```
let canvas = document.getElementById("myCanvas");
let ctx = canvas.getContext("2d");
ctx.font = "30px Arial";
ctx.fillText("Hello World",10,50);
```

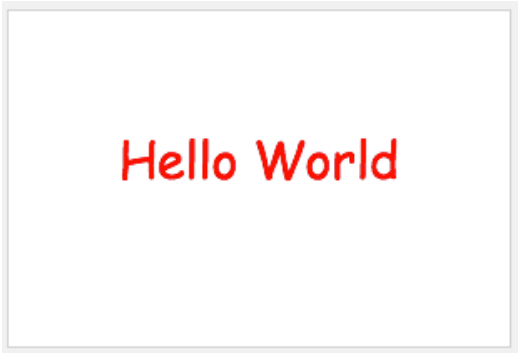

```
let canvas = document.getElementById("myCanvas");
let ctx = canvas.getContext("2d");
ctx.font = "30px Arial";
ctx.strokeText("Hello World",10,50);
```

Scripted graphics

To add color and center text:

- `fillStyle`
- `textAlign` - “start”, “end”, “right”, “center”, “left”

```
let canvas = document.getElementById("myCanvas");
let ctx = canvas.getContext("2d");
ctx.font = "30px Comic Sans MS";
ctx.fillStyle = "red";
ctx.textAlign = "center";
ctx.fillText("Hello World", canvas.width/2, canvas.height/2);
```



Scripted graphics

To draw an image on a canvas, use the following method:

- `drawImage(`*`img,x,y`*`)` - to position the image on the canvas;
- `drawImage(`*`img,x,y,width,height`*`)` - to position the image on the canvas, and specify width and height of the image;
- `drawImage(`*`img,sx,sy,swidth,sheight,x,y,width,height`*`)` - to clip the image and position the clipped part on the canvas; *`sx`* specifies the x coordinate where to start clipping; *`sy`* specifies the y coordinate where to start clipping; *`swidth`* specifies the width of the clipped image; *`sheight`* specifies the height of the clipped image.

Scripted graphics

```
<!DOCTYPE html>
<html>
<body>

<p>Canvas:</p>
<canvas id="myCanvas" width="600" height="500" style="border:1px solid #d3d3d3;">
Your browser does not support the HTML5 canvas tag.
</canvas>

<script>
let img = new Image();
img.width="110";
img.height="120";
img.src="https://www.icastellidelledonne.it/wp-content/uploads/elementor/thumbs/
castello_torrechiara-o7yotgdfkvmlby1s9wjceanm82nxb61uenq589aebk.jpg";
window.onload = function() {
    let c = document.getElementById("myCanvas");
    let ctx = c.getContext("2d");
    ctx.drawImage(img, 30, 40, 1000, 1200, 10, 10, 1000, 1200);
};
</script>

</body>
</html>
```

TypeScript

TypeScript for JavaScript programmers

TypeScript offers all of JavaScript's features, and an additional layer on top of these: **TypeScript's type system**.

For example, JavaScript provides language primitives like string and number, but it does not check that you have consistently assigned these. TypeScript does.

This means that your existing working JavaScript code is also TypeScript code.

The main benefit of TypeScript is that it can **highlight unexpected behavior** in your code, lowering the chance of bugs.

TypeScript tooling

- Install TypeScript
- Write your TypeScript code and save it to a .ts file
- Compile the .ts file using tsc and get the resulting .js file
- Include the .js file into your web app (like any other .js file)

https://www.typescriptlang.org/docs/handbook/typescript-tooling-in-5-minutes.html

Types by inference

TypeScript knows the JavaScript language and will generate types for you in many cases.

For example, in creating a variable and assigning it to a particular value, TypeScript will use the value as its type.

```
let helloWorld = "Hello World";
```

```
let helloWorld: string
```

Defining types

Interface declaration:

```
interface User {
  name: string;
  id: number;
}
```

Variable declaration:

```
const user: User = {
  name: "Hayes",
  id: 0,
};
```

https://ultimatecourses.com/blog/classes-vs-interfaces-in-typescript

An alternative to `interface` is `type`, but interface should be preferred.

Defining types

If you provide an object that does not match the declared interface, TypeScrypt warns you:

```
interface User {
  name: string;
  id: number;
}

const user: User = {
  username: "Hayes",
Type '{ username: string; id: number; }' is not assignable to type 'User'.
  Object literal may only specify known properties, and 'username' does
not exist in type 'User'.
  id: 0,
};
```

Defining types

An interface declaration can be used with classes:

```
interface User {
  name: string;
  id: number;
}

class UserAccount {
  name: string;
  id: number;

  constructor(name: string, id: number) {
    this.name = name;
    this.id = id;
  }
}

const user: User = new UserAccount("Murphy", 1);
```

Defining types

Interfaces can be used to annotate parameters and functions' returned values:

```
function getAdminUser(): User {
  //...
}

function deleteUser(user: User) {
  // ...
}
```

Defining types

Primitive types available in JavaScript: boolean, bigint, null, number, string, symbol, undefined.

TypeScript extends this list with a few more, such as:

- any
- unknown
- never
- void
- undefined

Composing types

With a union, it is possible to declare that a type can have one of many values.

```
type MyBool = true | false;
type WindowStates = "open" | "closed" | "minimized";
type LockStates = "locked" | "unlocked";
type PositiveOddNumbersUnderTen = 1 | 3 | 5 | 7 | 9;
```

With unions, it is also possible to handle different types.

```
function getLength(obj: string | string[]) {
  return obj.length;
}
```

Composing types

To learn the type of a variable, use `typeof`.

```
function wrapInArray(obj: string | string[]) {
  if (typeof obj === "string") {
    return [obj];
 }
  return obj;
}
```

Composing types

TypeScript offers **generics**, like C++:

```
type StringArray = Array<string>;
type NumberArray = Array<number>;
type ObjectWithNameArray = Array<{ name: string }>;
```

```
interface Backpack<Type> {
  add: (obj: Type) => void;
  get: () => Type;
}
declare const backpack: Backpack<string>;
const object = backpack.get();
backpack.add(23);
Argument of type 'number' is not assignable to parameter of type
'string'.
```

Structural type system

In TypeScript, type checking focuses on the *shape* that values have. If two object have the same shape, they are considered to be of the same type.

```
interface Point {
  x: number;
  y: number;
}

function logPoint(p: Point) {
  console.log(`${p.x}, ${p.y}`);
}

// logs "12, 26"
const point = { x: 12, y: 26 };
logPoint(point);
```

Here, point is never declared to be an instance of a Point, but its shape is that of a Point, so the code passes.

Frameworks and Development Tools

Frameworks

JavaScript frameworks serve as a skeleton for **single page apps**, allow developers to worry less about code structure or maintenance while focusing on creation of complex interface elements, and expand opportunities of JavaScript and plain HTML.

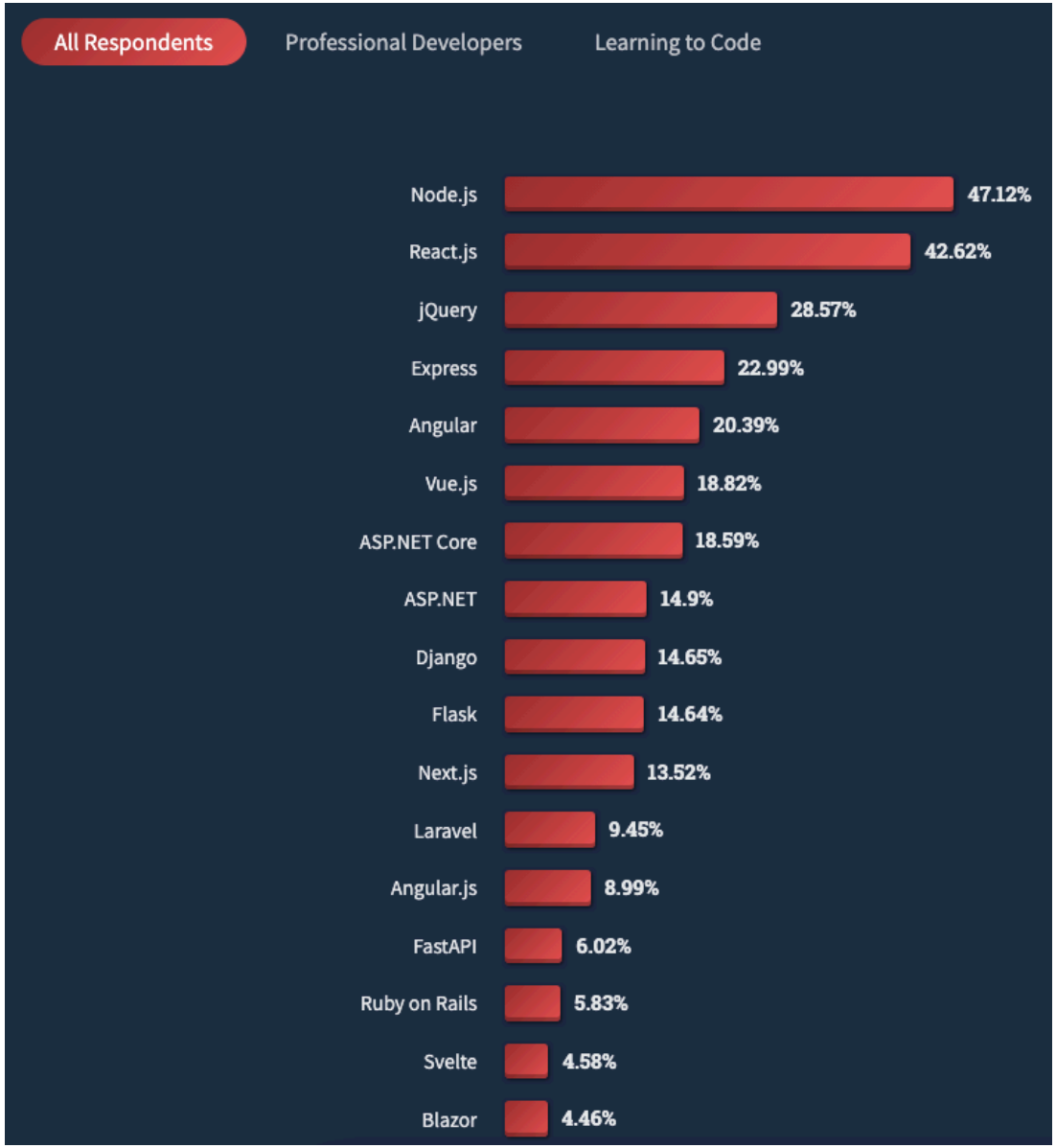
The advantages of using JavaScript frameworks:

1. **Efficiency** - projects that used to take months and hundreds of lines of code now can be achieved much faster with well-structured prebuilt patterns and functions.
2. **Safety** - top JavaScript frameworks are supported by large communities where members and users also act as testers.
3. **Cost** - most frameworks are open source and free. Since they help programmers to build custom solutions faster, the ultimate price for web app will be lower.

Frameworks

Stack Overflow Survey 2022

https://survey.stackoverflow.co/2022/

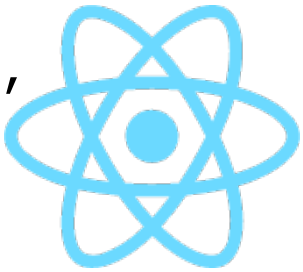


Frameworks

Angular comes with a long list of features that enable building everything, ranging from web to desktop and mobile.

https://angular.io/

Frameworks



React stands behind user interfaces of Facebook and Instagram, showing its efficiency within dynamic high-traffic applications.

Requires a higher learning curve, but makes app development straightforward and easy-to-understand.

Furthermore it can be a perfect fit for complex, high-load and awesome software solutions.

https://reactjs.org/

React was the preferred framework for the students of this course, in the last three years!

Frameworks

Back in 2015 **Ember** was called the best JavaScript framework for web application, leaving behind React and AngularJS (Angular's predecessor).

Today it boasts a huge online community, regular updates and wide appliance of JavaScript best practices to guarantee ultimate experience right out of the box.

Among the top users are Chipotle, Blue Apron, Nordstrom, Kickstarter, LinkedIn, Netflix and many others.

https://www.emberjs.com/

Frameworks

Vue was introduced in 2016 and it took the best from Ember, React and Angular, putting all that into a handy package. It is proved to be faster and leaner, comparing to React and Angular.

Vue is a better choice for quick development of cross-platform solutions. It can become a firm basis for high-end single page apps and beneficial solution to those cases, when performance is put ahead of good code organization or app structure.

https://vuejs.org/

Frameworks

Meteor is among the most popular JavaScript frameworks, with tons of features for back-end development, front-end rendering, database management and business logic.

This full-stack platform enables fast development of end-to-end web and mobile applications in pure JavaScript. Due to modular structure all the packages and libraries can be used at pace. In terms of performance, all the changes in the database are immediately transmitted to the UI and in conversely with no evident time losses caused by different languages or server response time.

It is of current usage in real-time application development for business companies like Mazda, IKEA, Honeywell and many others.

https://www.meteor.com/

JavaScript IDE and editors

WebStorm is a powerful IDE for advanced JavaScript development. It offers support for various frameworks and stylesheet languages, both web and server, mobile and desktop.

https://www.jetbrains.com/webstorm/ (free 30-days trial)

Atom from GitHub team is the number 1 choice for lots of people. It's an easily customizable text editor that comes with multiple features right out of the box. Atom includes embedded package manager, smart auto-completion, file system browser, cross-platform support, and some other useful functions.

https://atom.io/ (open source)

JavaScript IDE and editors

Visual Studio Code is backed by Microsoft and complete with the ultimate support for TypeScript right out of the box. It offers smart completions and syntax highlighting with IntelliSense, debugging right from the editor, built-in Git commands, version control, and so on.

https://code.visualstudio.com/ (open source)

Brackets is a lightweight text editor. It is mainly focused on visual tools and preprocessor support, to make it easier for you to design in the browser. Brackets comes with convenient real-time preview and powerful inline editors.

http://brackets.io/ (open source)

Other tools

Mobile-first sites development tools:
Bootstrap

Documentation tools:
Swagger, JSDog, JGrouseDog, YUIDoc, Docco

Testing tools:
Jasmine, Mocha, PhantomJS, Protractor

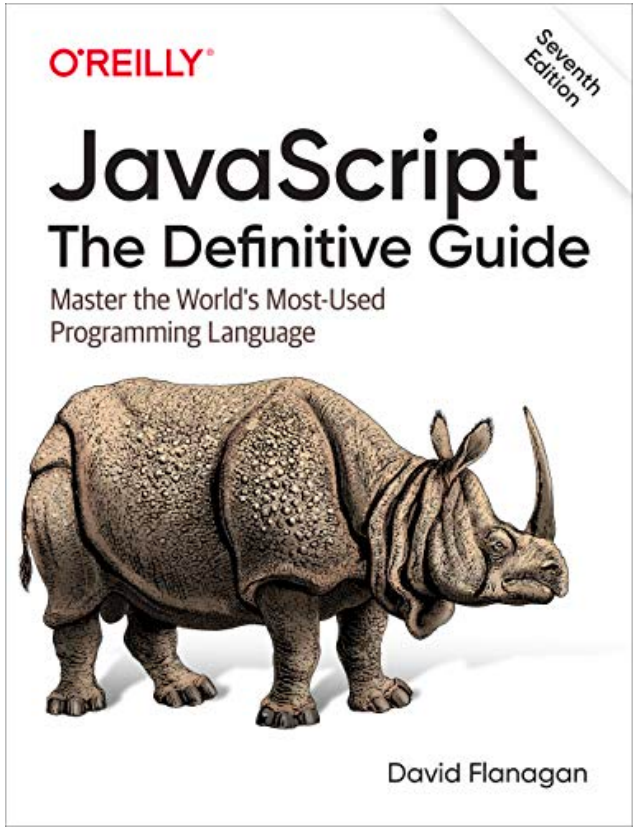
Debugging tools:
JavaScript Debugger, Chrome Dev Tools, ng-inspector, Augury

Security tools:
Snyk, Node Security Project, RetireJS, Gemnasium, OSSindex

Code optimization and analysis:
JSLint, JSHint, ESLint, Flow

References

D. Flanagan
JavaScript - The Definitive Guide
ed. O'Reilly, 2020



http://www.w3schools.com/js/default.asp

https://survey.stackoverflow.co/2022/