document.getElementById("demo").innerHTML = "Hello JavaScript";

JavaScript accepts both double and single quotes:

JavaScript Can Change HTML Attribute Values

In this example JavaScript changes the value of the src (source) attribute of an <img> tag:

The light Bulb project

JavaScript Can Change HTML Styles (CSS)

Changing the style of an HTML element, is a variant of changing an HTML attribute:

Example

document.getElementById("demo").style.fontSize = "35px";

<!DOCTYPE html>

<html>

<body>

<h2>What Can JavaScript Do?</h2>

<p>JavaScript can change HTML attribute values.</p>

<p>In this case JavaScript changes the value of the src (source) attribute of an image.</p>

<button onclick="document.getElementById('myImage').src='pic\_bulbon.gif'">Turn on the light</button>

<img id="myImage" src="pic\_bulboff.gif" style="width:100px">

<button onclick="document.getElementById('myImage').src='pic\_bulboff.gif'">Turn off the light</button>

</body>

</html>

JavaScript Can Hide HTML Elements

Hiding HTML elements can be done by changing the display style:

Example

document.getElementById("demo").style.display = "none";

or u can show html elements

document.getElementById("demo").style.display = "block";

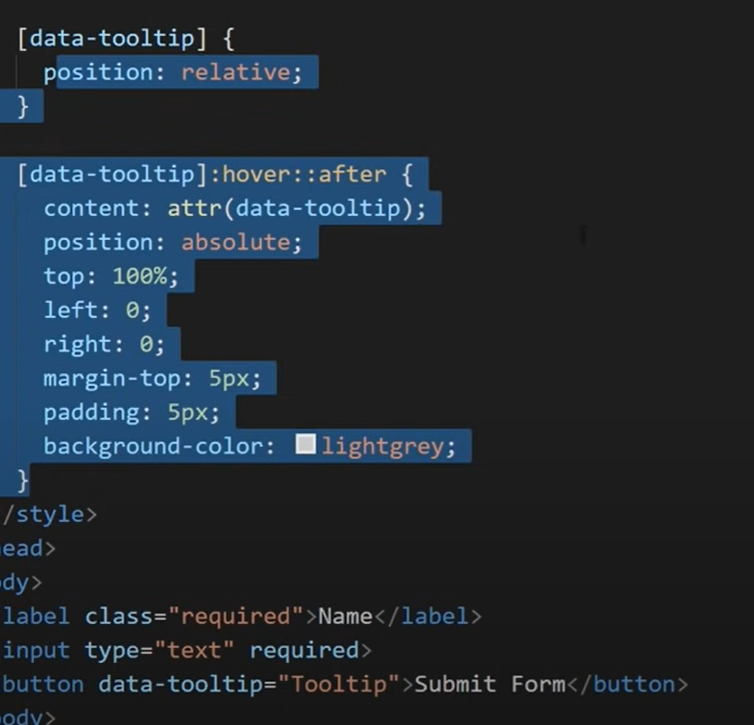
ECMA-262 is the official name of the standard. ECMAScript is the official name of the language.

In HTML, the span tag is a generic inline container element. You use this element to wrap sections of text for styling purposes or to add attributes to a section of text without creating a new line of content.

::before. In CSS, ::before creates a pseudo-element that is the first child of the selected element. It is often used to add cosmetic content to an element with the content property. It is inline by default.

Pseudoelements-allows you to create elements from css

U cant have pseudo elements inside of things that dont have content



TOGGLE SWITCH COMPLEX

<!-- Rectangular switch -->  
<label class="switch">  
  <input type="checkbox">  
  <span class="slider"></span>  
</label>  
  
<!-- Rounded switch -->  
<label class="switch">  
  <input type="checkbox">  
  <span class="slider round"></span>  
</label>

/\* The switch - the box around the slider \*/  
.switch {  
  position: relative;  
  display: inline-block;  
  width: 60px;  
  height: 34px;  
}  
  
/\* Hide default HTML checkbox \*/  
.switch input {  
  opacity: 0;  
  width: 0;  
  height: 0;  
}  
  
/\* The slider \*/  
.slider {  
  position: absolute;  
  cursor: pointer;  
  top: 0;  
  left: 0;  
  right: 0;  
  bottom: 0;  
  background-color: #ccc;  
  -webkit-transition: .4s;  
  transition: .4s;  
}  
  
.slider:before {  
  position: absolute;  
  content: "";  
  height: 26px;  
  width: 26px;  
  left: 4px;  
  bottom: 4px;  
  background-color: white;  
  -webkit-transition: .4s;  
  transition: .4s;  
}  
  
input:checked + .slider {  
  background-color: #2196F3;  
}  
  
input:focus + .slider {  
  box-shadow: 0 0 1px #2196F3;  
}  
  
input:checked + .slider:before {  
  -webkit-transform: translateX(26px);  
  -ms-transform: translateX(26px);  
  transform: translateX(26px);  
}  
  
/\* Rounded sliders \*/  
.slider.round {  
  border-radius: 34px;  
}  
  
.slider.round:before {  
  border-radius: 50%;  
}

### **Example**

Select and style the first <p> element that are placed immediately after <div> elements:

div + p {  
  background-color: yellow;  
}

### **Example**

Set the height and width for all checked <input> elements:

input:checked {  
  height: 50px;  
  width: 50px;  
}

The :checked selector matches every checked <input> element (only for radio buttons and checkboxes) and <option> element.

### **Example**

Select and style an input field when it gets focus:

input:focus {  
  background-color: yellow;  
}

The :focus selector is used to select the element that has focus.

**Tip:** The :focus selector is allowed on elements that accept keyboard events or other user inputs.

input[type=text] {  
  width: 100px;  
  transition: ease-in-out, width .35s ease-in-out;  
}  
  
input[type=text]:focus {  
  width: 250px;  
}

Change Several Property Values

The following example adds a transition effect for both the width and height property, with a duration of 2 seconds for the width and 4 seconds for the height:

Example

div {  
  transition: width 2s, height 4s;  
}

[Try it Yourself »](https://www.w3schools.com/css/tryit.asp?filename=trycss3_transition2)

Transition + Transformation

The following example adds a transition effect to the transformation:

Example

div {  
  transition: width 2s, height 2s, transform 2s;  
}

<https://developer.mozilla.org/en-US/docs/Web/CSS/transform>

**Stacking context**

**Stacking context** is a three-dimensional conceptualization of HTML elements along an imaginary z-axis relative to the user, who is assumed to be facing the viewport or the webpage. HTML elements occupy this space in priority order based on element attributes.

The :focus CSS pseudo-class represents an element (such as a form input) that has received focus. It is generally triggered when the user clicks or taps on an element or selects it with the keyboard's Tab key.

The :focus CSS pseudo-class represents an element (such as a form input) that has received focus. It is generally triggered when the user clicks or taps on an element or selects it with the keyboard's Tab key.

Js positions: absolute- relatigve to the parent element

Relative-relativan svojoj prosloj poziciji

Fixed-relativan web browseru

*// create a new `Date` object*

**const** now = **new** **Date**();

*// get the current date and time as a string*

**const** currentDateTime = now.**toLocaleString**();

console.**log**(currentDateTime);

In the code above, we create a new Date object called now which contains the current date and time. Then we use the toLocaleString() method to convert the date and time to a human-readable string in the format "MM/DD/YYYY, HH:MM:SS AM/PM". We store this string in a variable called currentDateTime and then log it to the console.

RESPONSIVE WEB DESIGN

## Setting The Viewport

To create a responsive website, add the following <meta> tag to all your web pages:

Example

<meta name="viewport" content="width=device-width, initial-scale=1.0">

Ovo kontrolise pages dimensions and scaling

This will set the viewport of your page, which will give the browser instructions on how to control the page's dimensions and scaling.

Responsive Images

If the CSS width property is set to 100%, the image will be responsive and scale up and down:

<img src="img\_girl.jpg" **style="width:100%;"**>

Notice that in the example above, the image can be scaled up to be larger than its original size. A better solution, in many cases, will be to use the max-width property instead.

Using the max-width Property

If the max-width property is set to 100%, the image will scale down if it has to, but never scale up to be larger than its original size:

<img src="img\_girl.jpg" style="**max-width:100%;**height:auto;">

Show Different Images Depending on Browser Width

The HTML <picture> element allows you to define different images for different browser window sizes.

Example

<picture>  
  <source srcset="img\_smallflower.jpg" media="(max-width: 600px)">  
  <source srcset="img\_flowers.jpg" media="(max-width: 1500px)">  
  <source srcset="flowers.jpg">  
  <img src="img\_smallflower.jpg" alt="Flowers">  
</picture>

Responsive Text Size

The text size can be set with a "vw" unit, which means the "viewport width".

That way the text size will follow the size of the browser window:

Definition and Usage

The <picture> tag gives web developers more flexibility in specifying image resources.

The most common use of the <picture> element will be for art direction in responsive designs. Instead of having one image that is scaled up or down based on the viewport width, multiple images can be designed to more nicely fill the browser viewport.

The <picture> element contains two tags: one or more [<source>](https://www.w3schools.com/tags/tag_source.asp) tags and one [<img>](https://www.w3schools.com/tags/tag_img.asp) tag.

The browser will look for the first <source> element where the media query matches the current viewport width, and then it will display the proper image (specified in the srcset attribute). The <img> element is required as the last child of the <picture> element, as a fallback option if none of the source tags matches.

**Tip:** The <picture> element works "similar" to <video> and <audio>. You set up different sources, and the first source that fits the preferences is the one being used.

Viewport is the browser window size. 1vw = 1% of viewport width. If the viewport is 50cm wide, 1vw is 0.5cm.

Media Queries

In addition to resize text and images, it is also common to use media queries in responsive web pages.

With media queries you can define completely different styles for different browser sizes.

Example: resize the browser window to see that the three div elements below will display horizontally on large screens and stack vertically on small screens:

/\* Use a media query to add a breakpoint at 800px: \*/  
@media screen and (max-width: 800px) {  
  .left, .main, .right {  
    width: 100%; /\* The width is 100%, when the viewport is 800px or smaller \*/  
  }

Responsive web design uses only HTML and CSS.

The viewport is the user's visible area of a web page.

The viewport varies with the device, and will be smaller on a mobile phone than on a computer screen.

Before tablets and mobile phones, web pages were designed only for computer screens, and it was common for web pages to have a static design and a fixed size.

Then, when we started surfing the internet using tablets and mobile phones, fixed size web pages were too large to fit the viewport. To fix this, browsers on those devices scaled down the entire web page to fit the screen.

Setting The Viewport

HTML5 introduced a method to let web designers take control over the viewport, through the <meta> tag.

You should include the following <meta> viewport element in all your web pages:

<meta name="viewport" content="width=device-width, initial-scale=1.0">

This gives the browser instructions on how to control the page's dimensions and scaling.

The width=device-width part sets the width of the page to follow the screen-width of the device (which will vary depending on the device).

The initial-scale=1.0 part sets the initial zoom level when the page is first loaded by the browser.

1. **Do NOT use large fixed width elements -**For example, if an image is displayed at a width wider than the viewport it can cause the viewport to scroll horizontally. Remember to adjust this content to fit within the width of the viewport.
2. **Do NOT let the content rely on a particular viewport width to render well**
3. **Use CSS media queries to apply different styling for small and large screens** - Setting large absolute CSS widths for page elements will cause the element to be too wide for the viewport on a smaller device. Instead, consider using relative width values, such as width: 100%. Also, be careful of using large absolute positioning values. It may cause the element to fall outside the viewport on small devices.

## What is a Grid-View?

Many web pages are based on a grid-view, which means that the page is divided into columns:

Using a grid-view is very helpful when designing web pages. It makes it easier to place elements on the page.

A responsive grid-view often has 12 columns, and has a total width of 100%, and will shrink and expand as you resize the browser window.

Building a Responsive Grid-View

Lets start building a responsive grid-view.

First ensure that all HTML elements have the box-sizing property set to border-box. This makes sure that the padding and border are included in the total width and height of the elements.

Add the following code in your CSS:

\* {  
  box-sizing: border-box;  
}

Read more about the box-sizing property in our [CSS Box Sizing](https://www.w3schools.com/css/css3_box-sizing.asp) chapter.

The following example shows a simple responsive web page, with two columns:

25%

75%

.menu {  
  width: 25%;  
  float: left;  
}  
.main {  
  width: 75%;  
  float: left;  
}

The example above is fine if the web page only contains two columns.

However, we want to use a responsive grid-view with 12 columns, to have more control over the web page.

First we must calculate the percentage for one column: 100% / 12 columns = 8.33%.

Then we make one class for each of the 12 columns, class="col-" and a number defining how many columns the section should span:

CSS:

.col-1 {width: 8.33%;}  
.col-2 {width: 16.66%;}  
.col-3 {width: 25%;}  
.col-4 {width: 33.33%;}  
.col-5 {width: 41.66%;}  
.col-6 {width: 50%;}  
.col-7 {width: 58.33%;}  
.col-8 {width: 66.66%;}  
.col-9 {width: 75%;}  
.col-10 {width: 83.33%;}  
.col-11 {width: 91.66%;}  
.col-12 {width: 100%;}

All these columns should be floating to the left, and have a padding of 15px:

CSS:

[class\*="col-"] {  
  float: left;  
  padding: 15px;  
  border: 1px solid red;  
}

Each row should be wrapped in a <div>. The number of columns inside a row should always add up to 12:

HTML:

<div class="row">  
  <div class="col-3">...</div> <!-- 25% -->  
  <div class="col-9">...</div> <!-- 75% -->  
</div>

The columns inside a row are all floating to the left, and are therefore taken out of the flow of the page, and other elements will be placed as if the columns do not exist. To prevent this, we will add a style that clears the flow:

CSS:

.row::after {  
  content: "";  
  clear: both;  
  display: table;  
}

::after. In CSS, ::after creates a pseudo-element that is the last child of the selected element. It is often used to add cosmetic content to an element with the content property. It is inline by default.13. 10. 2023.

The clear CSS property sets whether an element must be moved below (cleared) floating elements that precede it. The clear property applies to floating and non-floating elements.

Setting display to table makes the element behave like a table. So you can make a replica of an HTML table without using the table element and corresponding elements such as tr and td . For example, in HTML, you can make a table with the <table> element and also a <div> , or any container of your choice.

The float CSS property places an element on the left or right side of its container, allowing text and inline elements to wrap around it. The element is removed from the normal flow of the page, though still remaining a part of the flow (in contrast to absolute positioning).

CSS Layout - float and clear

The CSS float property specifies how an element should float.

The CSS clear property specifies what elements can float beside the cleared element and on which side.

The float Property

The float property is used for positioning and formatting content e.g. let an image float left to the text in a container.

The float property can have one of the following values:

* left - The element floats to the left of its container
* right - The element floats to the right of its container
* none - The element does not float (will be displayed just where it occurs in the text). This is default
* inherit - The element inherits the float value of its parent
* In its simplest use, the float property can be used to wrap text around images.

Probaj float

## Example - Float Next To Each Other

Normally div elements will be displayed on top of each other. However, if we use float: left we can let elements float next to each other:

Znaci float cini da ono sto je block se ponasa kao inline? A display ostaje block

CSS Layout - clear and clearfix

The clear Property

When we use the float property, and we want the next element below (not on right or left), we will have to use the clear property.

The clear property specifies what should happen with the element that is next to a floating element.

The clear property can have one of the following values:

* none - The element is not pushed below left or right floated elements. This is default
* left - The element is pushed below left floated elements
* right - The element is pushed below right floated elements
* both - The element is pushed below both left and right floated elements
* inherit - The element inherits the clear value from its parent

left I right od njega ili float:right float:left

When clearing floats, you should match the clear to the float: If an element is floated to the left, then you should clear to the left. Your floated element will continue to float, but the cleared element will appear below it on the web page.

Example

This example clears the float to the left. Here, it means that the <div2> element is pushed below the left floated <div1> element:

div1 {  
  float: left;  
}  
  
div2 {  
  clear: left;  
}

## The clearfix Hack

If a floated element is taller than the containing element, it will "overflow" outside of its container. We can then add a clearfix hack to solve this problem:

.clearfix {  
  overflow: auto;  
}

Ali kome se daje ovaj overflow parent ili childu, postaje naporno

The overflow: auto clearfix works well as long as you are able to keep control of your margins and padding (else you might see scrollbars). The **new, modern clearfix hack** however, is safer to use, and the following code is used for most webpages:

Example

.clearfix::after {  
  content: "";  
  clear: both;  
  display: table;  
}

Nije mi uopste jasan ovaj kod

You have to set them individually.

I made a variable to point to the style object, because we are modifying more than one property and because with() { ... } is [considered harmful](http://www.yuiblog.com/blog/2006/04/11/with-statement-considered-harmful/).

Also, I set the 50% to both properties because of right to left assignment, and because assignment of this string to the two properties isn't a problem (make sure you understand how this works, e.g. var a = b = [] will set a and b to the **same** Array object, often not desired).

var elementStyle = document.getElementById("id").style;

elementStyle.position = "relative";

elementStyle.top = elementStyle.left = "50%";

gsap.to(.text (y:50, duration:1))

text-shadow: h-shadow v-shadow blur-radius color;

text-shadow: h-shadow v-shadow blur-radius color;

text-shadow: h-shadow v-shadow blur-radius color;

text-shadow: text-shadow: h-shadow v-shadow blur-radiusza

margin auto da radi

1. The element must be block-level, e.g. display: block or display: table
2. The element must not float
3. The element must not have a fixed or absolute position1

Off the top of other people's heads:

1. The element must have a width that is not auto2

Sta susta su bese events in javascript

For example, a function can be called when an **event** occurs, like when the user clicks a button

Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both. <button type="button" onclick="myFunction()">Try it</button>

4 11 4 27

Nije mi bas jasan odnos html-a i js-a gde su dostupni jedno drugom

Placing scripts at the bottom of the <body> element improves the display speed, because script interpretation slows down the display. H

External scripts are practical when the same code is used in many different web pages.

<script src="myScript.js"></script>

You can place an external script reference in <head> or <body> as you like.

The script will behave as if it was located exactly where the <script> tag is located.

## External JavaScript Advantages

* Cached JavaScript files can speed up page loads
* o add several script files to one page  - use several script tags:

### **Example**

* <script src="myScript1.js"></script>  
  <script src="myScript2.js"></script>

## External References

An external script can be referenced in 3 different ways:

* With a full URL (a full web address)
* With a file path (like /js/)
* Without any path

This example uses a **full URL** to link to myScript.js:

### **Example**

<script src="https://www.w3schools.com/js/myScript.js"></script>

Kako without any path ne razumem

This example uses no path to link to myScript.js:

### **Example**

<script src="myScript.js"></script>

a nopath je dakle kada se fajl nalazi u istom folderu

JavaScript Display Possibilities

JavaScript can "display" data in different ways:

* Writing into an HTML element, using innerHTML.
* Writing into the HTML output using document.write().
* Writing into an alert box, using window.alert().
* Writing into the browser console, using console.log().

document.getElementById("demo").innerHTML = 5 + 6;

izaci ce nam 11

Changing the innerHTML property of an HTML element is a common way to display data in HTML. Dakle innerHTML je property

<p>Never call document.write after the document has finished loading.

It will overwrite the whole document.</p>

<script>

document.write(5 + 6);

</script>-shadow v-shadow b

Using document.write() after an HTML document is loaded, will **delete all existing HTML**: <h1>My First Web Page</h1>  
<p>My first paragraph.</p>  
  
<button type="button" onclick="document.write(5 + 6)">Try it</button> to znaci koristiti after its loaded

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>My First Web Page</h1>  
<p>My first paragraph.</p>  
  
<script>  
window.alert(5 + 6);  
</script>  
  
</body>  
</html>

Ovo ce immediately po loadingu pagea da izvrsi

You can skip the window keyword.

In JavaScript, the window object is the global scope object. This means that variables, properties, and methods by default belong to the window object. This also means that specifying the window keyword is optional:

## JavaScript Print

JavaScript does not have any print object or print methods.

You cannot access output devices from JavaScript.

The only exception is that you can call the window.print() method in the browser to print the content of the current window.

## JavaScript Programs

A **computer program** is a list of "instructions" to be "executed" by a computer.

In a programming language, these programming instructions are called **statements**.

A **JavaScript program** is a list of programming **statements**.

In HTML, JavaScript programs are executed by the web browser.

## Semicolons ;

Semicolons separate JavaScript statements.

Add a semicolon at the end of each executable statement:

When separated by semicolons, multiple statements on one line are allowed:

a = 5; b = 6; c = a + b;

A good practice is to put spaces around operators ( = + - \* / ):

For best readability, programmers often like to avoid code lines longer than 80 characters.

If a JavaScript statement does not fit on one line, the best place to break it is after an operator:

## JavaScript Code Blocks

JavaScript statements can be grouped together in code blocks, inside curly brackets {...}.

The purpose of code blocks is to define statements to be executed together.

|  |  |
| --- | --- |
| var | Declares a variable |
| let | Declares a block variable |
| const | Declares a block constant |
| try | Implements error handling to a block of statements |

JavaScript Values

The JavaScript syntax defines two types of values:

* Fixed values
* Variable values

Fixed values are called **Literals**.

Variable values are called **Variables** nisam znala za literale

## Js literals mogu biti num sa ili bez decimala ili tekst sa double ili single quotom JavaScript Expressions

An expression is a combination of values, variables, and operators, which computes to a value.

The computation is called an evaluation.

For example, 5 \* 10 evaluates to 50:

The values can be of various types, such as numbers and strings.

For example, "John" + " " + "Doe", evaluates to "John Doe": isto expression

Identifiers are JavaScript names.

Identifiers are used to name variables and keywords, and functions.

A JavaScript name must begin with:

* A letter (A-Z or a-z)
* A dollar sign ($)
* Or an underscore (\_)

Subsequent characters may be letters, digits, underscores, or dollar signs.

**Hyphens:**

first-name, last-name, master-card, inter-city.

Hyphens are not allowed in JavaScript. They are reserved for subtractions.

**Underscore:**

first\_name, last\_name, master\_card, inter\_city.

**Upper Camel Case (Pascal Case):**

FirstName, LastName, MasterCard, InterCity

**Lower Camel Case:**

## JavaScript Character Set

JavaScript uses the **Unicode** character set.

Unicode covers (almost) all the characters, punctuations, and symbols in the world.

Block comments se koriste za formal documentation

### **Variables are Containers for Storing Data**

JavaScript Variables can be declared in 4 ways:

* Automatically
* Using var
* Using let
* Using const

Kako automatically da declarujemo var

In this first example, x, y, and z are undeclared variables.

They are automatically declared when first used:

### **Example**

x = 5;  
y = 6;  
z = x + y;

da l moguce

The var keyword should only be used in code written for older browsers.

## When to Use var, let, or const?

1. Always declare variables

2. Always use const if the value should not be changed

3. Always use const if the type should not be changed (Arrays and Objects)

4. Only use let if you can't use const

5. Only use var if you MUST support old browsers.

## JavaScript Identifiers

All JavaScript **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

## The Assignment Operator

In JavaScript, the equal sign (=) is an "assignment" operator, not an "equal to" operator.

This is different from algebra. The following does not make sense in algebra:

x = x + 5

In JavaScript, however, it makes perfect sense: it assigns the value of x + 5 to x.

(It calculates the value of x + 5 and puts the result into x. The value of x is incremented by 5.)

## Note

The "equal to" operator is written like == in JavaScript.

## JavaScript Data Types

JavaScript variables can hold numbers like 100 and text values like "John Doe".

In programming, text values are called text strings.

JavaScript can handle many types of data, but for now, just think of numbers and strings.

After the declaration, the variable has no value (technically it is undefined).

To **assign** a value to the variable, use the equal sign:

<p id="demo"></p>  
  
<script>  
let carName = "Volvo";  
document.getElementById("demo").innerHTML = carName;  
</script>

## Note

It's a good programming practice to declare all variables at the beginning of a script.

Ali to samo ako su var kako let deklarovat van f akoo nije globalna?

let person = "John Doe", carName = "Volvo", price = 200;

one statement many variables

## Re-Declaring JavaScript Variables

If you re-declare a JavaScript variable declared with var, it will not lose its value.

## Note

You cannot re-declare a variable declared with let or const.

This will not work:

let carName = "Volvo";  
let carName;

You can also add strings, but strings will be concatenated:

### **Example**

let x = "John" + " " + "Doe";

let x = "5" + 2 + 3; 523 a ja bih rekla 55

## Note

If you put a number in quotes, the rest of the numbers will be treated as strings, and concatenated.

Vremena se menjaju

let x = 2 + 3 + "5"; ovo radi onako kako ti zelis da radi tj mislis

## JavaScript Dollar Sign $

Since JavaScript treats a dollar sign as a letter, identifiers containing $ are valid variable names:

Using the dollar sign is not very common in JavaScript, but professional programmers often use it as an alias for the main function in a JavaScript library.

In the JavaScript library jQuery, for instance, the main function $ is used to select HTML elements. In jQuery $("p"); means "select all p elements".

Using the underscore is not very common in JavaScript, but a convention among professional programmers is to use it as an alias for "private (hidden)" variables.

Variables declared with let have **Block Scope**

Variables declared with let must be **Declared** before use

Variables declared with let cannot be **Redeclared** in the same scope

**Block Scope**.

JavaScript had **Global Scope** and **Function Scope**.

ES6 introduced the two new JavaScript keywords: let and const.

These two keywords provided **Block Scope** in JavaScript:

### **Example**

Variables declared inside a { } block cannot be accessed from outside the block:

{  
  let x = 2;  
}  
// x can NOT be used here

U cemu je razlika izmedju block scope and function scope-a

## Global Scope

Variables declared with the var always have **Global Scope**.

Variables declared with the var keyword can NOT have block scope:

You can not accidentally redeclare a variable declared with let.

With let you **can not** do this:

let x = "John Doe";  
  
let x = 0;

Variables defined with var **can** be redeclared.

Redeclaring a variable inside a block will not redeclare the variable outside the block:

## Difference Between var, let and const

|  |
| --- |
|  |
|  | Scope | Redeclare | Reassign | Hoisted | Binds this |
| var | No | Yes | Yes | Yes | Yes |
| let | Yes | No | Yes | No | No |
| const | Yes | No | No | No | No |

Sta znaci binds this

let and const have **block scope**.

let and const can not be **redeclared**.

let and const must be **declared** before use.

let and const does **not bind** to this.

let and const are **not hoisted**.

## What is Not Good?

var does not have to be declared.

Sta ovo znaci?

Je l to znaci ako napisemo x=10; da autoamtski se pretpostavlaja da je var a ne const

## Redeclaring

Redeclaring a JavaScript variable with var is allowed anywhere in a program:

With let, redeclaring a variable in the same block is NOT allowed:

### **Example**

var x = 2;   // Allowed  
let x = 3;   // Not allowed  
  
{  
let x = 2;   // Allowed  
let x = 3;   // Not allowed  
}  
  
{  
let x = 2;   // Allowed  
var x = 3;   // Not allowed  
}

### **Example**

var x = 2;   // Allowed  
let x = 3;   // Not allowed  
  
{  
let x = 2;   // Allowed  
let x = 3;   // Not allowed  
}  
  
{  
let x = 2;   // Allowed  
var x = 3;   // Not allowed  
}

Redeclaring a variable with let, in another block, IS allowed:

### **Example**

let x = 2;   // Allowed  
  
{  
let x = 3;   // Allowed  
}  
  
{  
let x = 4;    // Allowed  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_let_redeclare)

## Let Hoisting

Variables defined with var are **hoisted** to the top and can be initialized at any time.

Meaning: You can use the variable before it is declared:

This is OK:

carName = "Volvo";  
var carName;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_let_hoisting_var)

Variables defined with let are also hoisted to the top of the block, but not initialized.

Meaning: Using a let variable before it is declared will result in a ReferenceError:

Initialization is the process of assigning a value to the Variable.

## JavaScript Hoisting

With **let**, you cannot use a variable before it is declared.

ReferenceError: Cannot access 'carName' before initialization

Variables defined with const cannot be **Redeclared**

Variables defined with const cannot be **Reassigned**

Variables defined with const have **Block Scope**

JavaScript const variables must be assigned a value when they are declared:

When to use JavaScript const?

**Always declare a variable with const when you know that the value should not be changed.**

Use const when you declare:

* A new Array
* A new Object
* A new Function
* A new RegExp

Constant Objects and Arrays

The keyword const is a little misleading.

It does not define a constant value. It defines a constant reference to a value.

Because of this you can NOT:

* Reassign a constant value
* Reassign a constant array
* Reassign a constant object

But you CAN:

* Change the elements of constant array
* Change the properties of constant object

Vise je konstantno ono s leve strane u deklaraciji nego ono s leve strane

// You can create a constant array:  
const cars = ["Saab", "Volvo", "BMW"];  
  
// You can change an element:  
cars[0] = "Toyota";  
  
// You can add an element:  
cars.push("Audi");

But you can NOT reassign the array:

### **Example**

const cars = ["Saab", "Volvo", "BMW"];  
  
cars = ["Toyota", "Volvo", "Audi"];    // ERROR

## Constant Objects

You can change the properties of a constant object:

### **Example**

// You can create a const object:  
const car = {type:"Fiat", model:"500", color:"white"};  
  
// You can change a property:  
car.color = "red";  
  
// You can add a property:  
car.owner = "Johnson";

But you can NOT reassign the object:

### **Example**

const car = {type:"Fiat", model:"500", color:"white"};  
  
car = {type:"Volvo", model:"EX60", color:"red"};    // ERROR

## Block Scope

Declaring a variable with const is similar to let when it comes to **Block Scope**.

The x declared in the block, in this example, is not the same as the x declared outside the block:

### **Example**

const x = 10;  
// Here x is 10  
  
{  
const x = 2;  
// Here x is 2  
}  
  
// Here x is 10

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_const)

## Redeclaring

Redeclaring a JavaScript var variable is allowed anywhere in a program:

### **Example**

var x = 2;     // Allowed  
var x = 3;     // Allowed  
x = 4;         // Allowed

Redeclaring an existing var or let variable to const, in the same scope, is not allowed:

Redeclaring an existing var or let variable to const, in the same scope, is not allowed:

### **Example**

var x = 2;     // Allowed  
const x = 2;   // Not allowed  
  
{  
let x = 2;     // Allowed  
const x = 2;   // Not allowed  
}  
  
{  
const x = 2;   // Allowed  
const x = 2;   // Not allowed  
}

Reassigning an existing const variable, in the same scope, is not allowed:

Redeclaring a variable with const, in another scope, or in another block, is allowed:

### **Example**

const x = 2;       // Allowed  
  
{  
  const x = 3;   // Allowed  
}  
  
{  
  const x = 4;   // Allowed  
}

Variables defined with const are also hoisted to the top, but not initialized.

Meaning: Using a const variable before it is declared will result in a ReferenceError:

Types of JavaScript Operators

There are different types of JavaScript operators:

* Arithmetic Operators
* Assignment Operators
* Comparison Operators
* String Operators
* Logical Operators
* Bitwise Operators
* Ternary Operators
* Type Operators

|  |  |
| --- | --- |
| \*\* | Exponentiation ([ES2016](https://www.w3schools.com/js/js_2016.asp)) |

The **Addition Assignment Operator** (+=) adds a value to a variable.

Mozes komparisat i stringe a to se komparise alphabethically

Note that strings are compared alphabetically:

let text1 = "What a very ";  
text1 += "nice day";

When used on strings, the + operator is called the concatenation operator.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| && | logical and |
| || | logical or |
| ! | logical not |

## JavaScript Type Operators

|  |  |
| --- | --- |
| **Operator** | **Description** |
| typeof | Returns the type of a variable |
| instanceof | Returns true if an object is an instance of an object type |

## JavaScript Bitwise Operators

Bit operators work on 32 bits numbers.

Any numeric operand in the operation is converted into a 32 bit number. The result is converted back to a JavaScript number.

Wow

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| << | Zero fill left shift | Shifts left by pushing zeros in from the right and let the leftmost bits fall off | | | |
| >> | Signed right shift | Shifts right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off | | | |
| >>> | Zero fill right shift | Shifts right by pushing zeros in from the left, and let the rightmost bits | | | |
| & | AND | 5 & 1 | 0101 & 0001 | 0001 | 1 |
| | | OR | 5 | 1 | 0101 | 0001 | 0101 | 5 |
| ~ | NOT | ~ 5 | ~0101 | 1010 | 10 |
| ^ | XOR | 5 ^ 1 | 0101 ^ 0001 | 0100 | 4 |
| << | left shift | 5 << 1 | 0101 << 1 | 1010 | 10 |
| >> | right shift | 5 >> 1 | 0101 >> 1 | 0010 | 2 |
| >>> | unsigned right shift | 5 >>> 1 | 0101 >>> 1 | 0010 | 2 |

A bit (binary digit) is the smallest unit of data that a computer can process and store. A bit is always in one of two physical states, similar to an on/off light switch. The state is represented by a single binary value, usually a 0 or 1. However, the state might also be represented by yes/no, on/off or true/false.

The examples above uses 4 bits unsigned examples. But JavaScript uses 32-bit signed numbers.  
Because of this, in JavaScript, ~ 5 will not return 10. It will return -6.  
~00000000000000000000000000000101 will return 11111111111111111111111111111010

Bitwise operators are fully described in the [**JS Bitwise**](https://www.w3schools.com/js/js_bitwise.asp) chapter.

let x = 100 + 50;

numbers that are literals

let x = (100 + 50) \* a;

ovo u zagradi je expression

In arithmetic, the division of two integers produces a **quotient** and a **remainder**.

In mathematics, the result of a **modulo operation** is the **remainder** of an arithmetic division.

The **exponentiation** operator (\*\*) raises the first operand to the power of the second operand.

### **Example**

let x = 5;  
let z = x \*\* 2;

x \*\* y produces the same result as Math.pow(x,y):

## Operator Precedence

Operator precedence describes the order in which operations are performed in an arithmetic expression.

Isto kao u matematici

## Shift Assignment Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Same As** |
| <<= | x <<= y | x = x << y |
| >>= | x >>= y | x = x >> y |
| >>>= | x >>>= y | x = x >>> y |

Je l ovo ima smisla za bilo sta sto nije bitwise

## Bitwise Assignment Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Same As** |
| &= | x &= y | x = x & y |
| ^= | x ^= y | x = x ^ y |
| |= | x |= y | x = x | y |

## Logical Assignment Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Same As** |
| &&= | x &&= y | x = x && (x = y) |
| ||= | x ||= y | x = x || (x = y) |
| ??= | x ??= y | x = x ?? (x = y) |

Sta li je ovo sa dva upitnika

let text = "Hello"; text += " World";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_assign_plusequal2)

<script>

let x = 10;

x %= 5;

document.getElementById("demo").innerHTML = "Value of x is: " + x;

</script>

The **Left Shift Assignment Operator** left shifts a variable.

### **Left Shift Assignment Example**

let x = -100;  
x <<= 5;

ovo mora se nekad kasnije razumit

right shiftovat se moze varijabla koja je signed i koje je unsigned

sta je signed sta je unsigned varijabla

## The >>= Operator

The **Right Shift Assignment Operator** right shifts a variable (signed).

### **Right Shift Assignment Example**

let x = -100;  
x >>= 5;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_assign_right_shift)

## The >>>= Operator

The **Unsigned Right Shift Assignment Operator** right shifts a variable (unsigned).

### **Unsigned Right Shift Assignment Example**

let x = -100;  
x >>>= 5;

n computing, signedness is a property of data types representing numbers in computer programs. A numeric variable is signed if it can represent both positive and negative numbers, and unsigned if it can only represent non-negative numbers (zero or positive numbers).

lur-radius color;

Initialization is the process of assigning a value to the Variable. Every programming language has its own method of initializing the variable. If the value is not assigned to the Variable, then the process is only called a Declaration.

## Grid Layout

The CSS Grid Layout Module offers a grid-based layout system, with rows and columns, making it easier to design web pages without having to use floats and positioning.

## Display Property

An HTML element becomes a grid container when its display property is set to grid or inline-grid.

All direct children of the grid container automatically become grid items.

## Grid Gaps

The spaces between each column/row are called gaps.

You can adjust the gap size by using one of the following properties:

* column-gap
* row-gap
* gap

daju se containeru ove properties

## Grid Lines

The lines between columns are called column lines.

The lines between rows are called row lines.

Refer to line numbers when placing a grid item in a grid container:

Example

Place a grid item at column line 1, and let it end on column line 3:

.item1 {  
grid-column-start: 1;  
  grid-column-end: 3;  
}

Place a grid item at row line 1, and let it end on row line 3:

.item1 {  
grid-row-start: 1;  
  grid-row-end: 3;  
}

All CSS Grid Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| [column-gap](https://www.w3schools.com/cssref/css3_pr_column-gap.asp) | Specifies the gap between the columns |
| [gap](https://www.w3schools.com/cssref/css3_pr_gap.asp) | A shorthand property for the *row-gap* and the *column-gap* properties |
| [grid](https://www.w3schools.com/cssref/pr_grid.asp) | A shorthand property for the *grid-template-rows, grid-template-columns, grid-template-areas, grid-auto-rows, grid-auto-columns*, and the *grid-auto-flow* properties |
| [grid-area](https://www.w3schools.com/cssref/pr_grid-area.asp) | Either specifies a name for the grid item, or this property is a shorthand property for the *grid-row-start*, *grid-column-start*, *grid-row-end*, and *grid-column-end* properties |
| [grid-auto-columns](https://www.w3schools.com/cssref/pr_grid-auto-columns.asp) | Specifies a default column size |
| [grid-auto-flow](https://www.w3schools.com/cssref/pr_grid-auto-flow.asp) | Specifies how auto-placed items are inserted in the grid |
| [grid-auto-rows](https://www.w3schools.com/cssref/pr_grid-auto-rows.asp) | Specifies a default row size |
| [grid-column](https://www.w3schools.com/cssref/pr_grid-column.asp) | A shorthand property for the *grid-column-start* and the *grid-column-end* properties |
| [grid-column-end](https://www.w3schools.com/cssref/pr_grid-column-end.asp) | Specifies where to end the grid item |
| [grid-column-gap](https://www.w3schools.com/cssref/pr_grid-column-gap.asp) | Specifies the size of the gap between columns |
| [grid-column-start](https://www.w3schools.com/cssref/pr_grid-column-start.asp) | Specifies where to start the grid item |
| [grid-gap](https://www.w3schools.com/cssref/pr_grid-gap.asp) | A shorthand property for the *grid-row-gap* and *grid-column-gap* properties |
| [grid-row](https://www.w3schools.com/cssref/pr_grid-row.asp) | A shorthand property for the *grid-row-start* and the *grid-row-end* properties |
| [grid-row-end](https://www.w3schools.com/cssref/pr_grid-row-end.asp) | Specifies where to end the grid item |
| [grid-row-gap](https://www.w3schools.com/cssref/pr_grid-row-gap.asp) | Specifies the size of the gap between rows |
| [grid-row-start](https://www.w3schools.com/cssref/pr_grid-row-start.asp) | Specifies where to start the grid item |
| [grid-template](https://www.w3schools.com/cssref/pr_grid-template.asp) | A shorthand property for the *grid-template-rows*, *grid-template-columns* and *grid-areas* properties |
| [grid-template-areas](https://www.w3schools.com/cssref/pr_grid-template-areas.asp) | Specifies how to display columns and rows, using named grid items |
| [grid-template-columns](https://www.w3schools.com/cssref/pr_grid-template-columns.asp) | Specifies the size of the columns, and how many columns in a grid layout |
| [grid-template-rows](https://www.w3schools.com/cssref/pr_grid-template-rows.asp) | Specifies the size of the rows in a grid layout |
| [row-gap](https://www.w3schools.com/cssref/css3_pr_row-gap.asp) | Specifies the gap between the grid rows |

## The grid-template-columns Property

The grid-template-columns property defines the number of columns in your grid layout, and it can define the width of each column.

The value is a space-separated-list, where each value defines the width of the respective column.

If you want your grid layout to contain 4 columns, specify the width of the 4 columns, or "auto" if all columns should have the same width.

**Note:** If you have more than 4 items in a 4 columns grid, the grid will automatically add a new row to put the items in.

## The grid-template-rows Property

The grid-template-rows property defines the height of each row.

## The justify-content Property

The justify-content property is used to align the whole grid inside the container. **Note:** The grid's total width has to be less than the container's width for the justify-content property to have any effect.

## The align-content Property

The align-content property is used to vertically align the whole grid inside the container.

The grid's total height has to be less than the container's height for the align-content property to have any effect.

Alignt content i justify content se dakle daju parent elementu u cssu

Prelazimo na grid items

## Child Elements (Items)

A grid container contains grid items.

By default, a container has one grid item for each column, in each row, but you can style the grid items so that they will span multiple columns and/or rows.

19:56

## The grid-column Property:

The grid-column property defines on which column(s) to place an item.

You define where the item will start, and where the item will end.

**Note:** The grid-column property is a shorthand property for the grid-column-start and the grid-column-end properties.

To place an item, you can refer to line numbers, or use the keyword "span" to define how many columns the item will span.

Example

Make "item1" start on column 1 and end before column 5:

.item1 {  
  grid-column: 1 / 5;  
}

Example

Make "item1" start on column 1 and span 3 columns:

.item1 {  
  grid-column: 1 / span 3;  
}

Example

Make "item1" start on row-line 1 and end on row-line 4:

.item1 {  
  grid-row: 1 / 4;  
}

To je za row a za column je before

## The grid-area Property

The grid-area property can be used as a shorthand property for the grid-row-start, grid-column-start, grid-row-end and the grid-column-end properties.

Example

Make "item8" start on row-line 1 and column-line 2, and end on row-line 5 and column line 6:

.item8 {  
  grid-area: 1 / 2 / 5 / 6;  
}

.item8 {  
  grid-area: 2 / 1 / span 2 / span 3;  
}

Naming Grid Items

The grid-area property can also be used to assign names to grid items.

Header

Named grid items can be referred to by the grid-template-areas property of the grid container.

Example

Item1 gets the name "myArea" and spans all five columns in a five columns grid layout:

.item1 {  
  grid-area: myArea;  
}  
.grid-container {  
  grid-template-areas: 'myArea myArea myArea myArea myArea';  
}

Each row is defined by apostrophes (' ')

The columns in each row is defined inside the apostrophes, separated by a space.

**Note:** A period sign represents a grid item with no name.

To define two rows, define the column of the second row inside another set of apostrophes:

Example

Make "item1" span two columns *and* two rows:

.grid-container {  
  grid-template-areas: 'myArea myArea . . .' 'myArea myArea . . .';  
}

## The Order of the Items

The Grid Layout allows us to position the items anywhere we like.

The first item in the HTML code does not have to appear as the first item in the grid.

What is implicit grid?

An implicit grid is an explicit grid with additional implicit tracks and line forms. This means if you have more grid items than cells in the grid or when a grid item is placed outside of the explicit grid, the grid container automatically generates grid tracks by adding grid lines to the grid.

You can re-arrange the order for certain screen sizes, by using media queries:

Example

@media only screen and (max-width: 500px) {  
  .item1 { grid-area: 1 / span 3 / 2 / 4; }  
  .item2 { grid-area: 3 / 3 / 4 / 4; }  
  .item3 { grid-area: 2 / 1 / 3 / 2; }  
  .item4 { grid-area: 2 / 2 / span 2 / 3; }  
  .item5 { grid-area: 3 / 1 / 4 / 2; }  
  .item6 { grid-area: 2 / 3 / 3 / 4; }  
}

CSS grid-auto-flow Property

Insert auto-placed items column by column:

.grid-container {  
  display: grid;  
  grid-auto-flow: column;  
}

|  |  |  |
| --- | --- | --- |
| **Value** | **Description** | **Demo** |
| row | Default value. Places items by filling each row | [Demo ❯](https://www.w3schools.com/cssref/playdemo.php?filename=playcss_grid-auto-flow) |
| column | Places items by filling each column | [Demo ❯](https://www.w3schools.com/cssref/playdemo.php?filename=playcss_grid-auto-flow&preval=column) |
| dense | Place items to fill any holes in the grid | [Demo ❯](https://www.w3schools.com/cssref/playdemo.php?filename=playcss_grid-auto-flow&preval=dense) |
| row dense | Places items by filling each row, and fill any holes in the grid | [Demo ❯](https://www.w3schools.com/cssref/playdemo.php?filename=playcss_grid-auto-flow&preval=row%20dense) |
| column dense | Places items by filling each column, and fill any holes in the grid |  |

<https://www.w3schools.com/cssref/playdemo.php?filename=playcss_grid-auto-flow>

With grid-template-columns: 1fr 1fr 1fr 1fr; we made four vertical tracks each with a width of 1fr. We can automate that by using the repeat() notation like this: grid-template-columns: repeat(4, 1fr);. The first argument specifies the number of repetitions, and the second a track list, which is repeated that number of times.

## Auto-fitting explicit grid

With an explicit grid, you can also auto-fit tracks into the grid container. The auto-fill keyword creates as many tracks as can fit into the grid container without causing the grid to overflow. With auto-fill, you can create a dynamic grid. Let's check it out the example.

Here, we will make each column 100px wide. We achieve this with the grid-template-columns, repeat, and auto-fill properties. Also, we will set the grid-gap to 20px so you can see the columns:

.item{

background-color: rgb(215, 181, 246);

text-align: center;

}

.grid {

display: grid;

grid-template-columns: repeat(auto-fill, 100px);

grid-template-rows: 100px 100px;

grid-gap: 20px;

}

Autofitting explicit grid nije lose

## Implicit grid

An implicit grid is an explicit grid with additional implicit tracks and line forms. This means if you have more grid items than cells in the grid or when a grid item is placed outside of the explicit grid, the grid container automatically generates grid tracks by adding grid lines to the grid.

Znaci implicit grid koristimo kada imamo more items than cells ili kada je velicina itema prevjelika

When you define a grid using CSS Grid Layout, you can specify the number of rows and columns that you want the grid to have.

The size and position of the grid items placed into the implicit grid are determined by the grid-auto-rows and grid-auto-columns properties, which specify the size of the rows and columns in the implicit grid. You can also use the grid-auto-flow property to control how the grid items are placed into the implicit grid, either in rows or columns.

.item{

background-color: rgb(215, 181, 246);

text-align: center;

}

.grid {

display: grid;

grid-template-columns: repeat(4, 1fr);

grid-template-rows: 100px 100px;

grid-gap: 20px;

max-width: 800px;

}

.item:first-child {

grid-column-start: -1;

}

.item:nth-child(2) {

grid-row-start: 4;

}

As you can see, there is quite a difference between the two outputs. The widths and heights of the implicit tracks are set automatically. They are only big enough to fit the placed grid items, but it’s possible to change this default behavior.

In previous topics, you've learned that the grid-auto-rows and grid-auto-columns properties give us control over the size of rows and columns in the grid. T

. This example will show you how to control the size of the implicit tracks:

.grid {

display: grid;

grid-template-columns: repeat(4, 1fr);

grid-template-rows: 100px 100px;

grid-gap: 20px;

grid-auto-columns: 200px;

grid-auto-rows: 60px;

max-width: 800px;

}

.item:first-child {

grid-column-start: -1;

}

.item:nth-child(2) {

grid-row-start: 4;

}

## Automatic Placement

Implicit tracks are added if the number of items exceeds the number of cells. The auto-placement algorithm places items by filling each row consecutively by default, adding new rows as necessary. You can also use the grid-auto-flow property to specify the flow of the grid.

Automatic placement of implicit grids in CSS is done using the grid-auto-flow property, which controls the placement of items that are not explicitly placed on the grid.

<https://www.w3schools.com/css/tryit.asp?filename=trycss_layout_float3>

<https://www.w3schools.com/css/tryit.asp?filename=trycss_layout_clearfix2>

<https://www.w3schools.com/css/css_float_clear.asp>

<https://www.w3schools.com/css/css_rwd_grid.asp>

<https://www.w3schools.com/howto/howto_css_switch.asp>

<https://www.w3schools.com/js/tryit.asp?filename=tryjs_intro_lightbulb>

In computing, signedness is a property of data types representing numbers in computer programs. A numeric variable is signed if it can represent both positive and negative numbers, and unsigned if it can only represent non-negative numbers (zero or positive numbers).

<https://www.w3schools.com/js/js_assignment.asp>

defer. This Boolean attribute is set to indicate to a browser that the script is meant to be executed after the document has been parsed, but before firing DOMContentLoaded . Scripts with the defer attribute will prevent the DOMContentLoaded event from firing until the script has loaded and finished evaluating.

