Thavorac C. - 2021-2022

HOW MUCH DO YOU KNOW ABOUT JAVASCRIPT?

Do you know this?

The best web browser in late 90s

Created by Brendan Eich in 1995, while he was working at Netscape

To compete with Internet Explorer

But it fail and recognized as a hated language

It was named as Mocha, then LiveScript and then Javascript (ECMA 262)

After Netscape handed JavaScript over to ECMA, the Mozilla foundation continued to develop JavaScript for the Firefox browser. Mozilla's latest version was 1.8.5. (Identical to ES5).

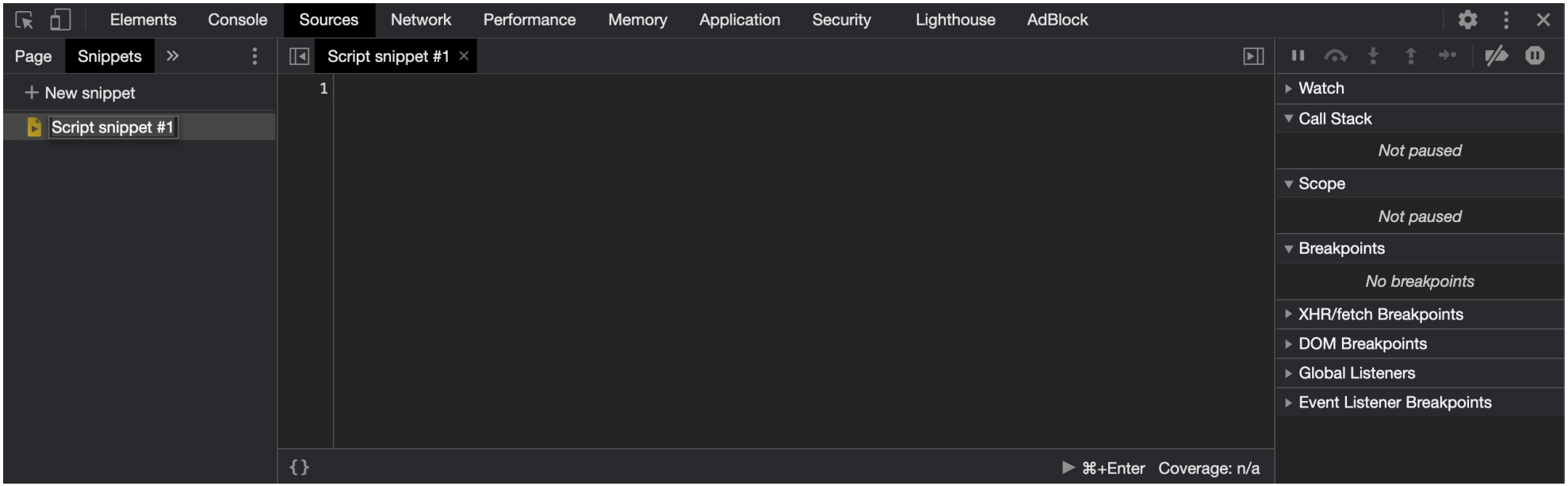
Source: https://medium.com/@_benaston/lesson-1a-the-history-of-javascript-8c1ce3bffb17

Year	ECMA	Browser
1995		JavaScript was invented by Brendan Eich
1996		Netscape 2 was released with JavaScript 1.0
1997		JavaScript became an ECMA standard (ECMA-262)
1997	ES1	ECMAScript 1 was released
1997	ES1	IE 4 was the first browser to support ES1
1998	ES2	ECMAScript 2 was released
1998		Netscape 42 was released with JavaScript 1.3
1999	ES2	IE 5 was the first browser to support ES2
1999	ES3	ECMAScript 3 was released
2000	ES3	IE 5.5 was the first browser to support ES3
2000		Netscape 62 was released with JavaScript 1.5
2000		Firefox 1 was released with JavaScript 1.5
2008	ES4	ECMAScript 4 was abandoned
2009	ES5	ECMAScript 5 was released
2011	ES5	IE 9 was the first browser to support ES5 *
2011	ES5	Firefox 4 was released with JavaScript 1.8.5

Year	ECMA	Browser
2012	ES5	Full support for ES5 in Safari 6
2012	ES5	Full support for ES5 in IE 10
2012	ES5	Full support for ES5 in Chrome 23
2013	ES5	Full support for ES5 in Firefox 21
2013	ES5	Full support for ES5 in Opera 15
2014	ES5	Full support for ES5 in all browsers
2015	ES6	ECMAScript 6 was released
2016	ES6	Full support for ES6 in Chrome 51
2016	ES6	Full support for ES6 in Opera 38
2016	ES6	Full support for ES6 in Edge 14
2016	ES6	Full support for ES6 in Safari 10
2015	ES6	Full support for ES6 in Firefox 52
2018	ES6	Full support for ES6 in browsers

WHAT HAPPENED WITHOUT JAVASCRIPT?

LET'S GET PRACTICE JAVASCRIPT



You can use google chrome's snippet to write javascript

WHAT JAVASCRIPT CAN DO?

- JavaScript Can Change HTML Content

  One of many JavaScript HTML methods is getElementById().

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

- JavaScript Can Change HTML Attribute Values
- JavaScript Can Change HTML Styles (CSS)

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

- JavaScript Can Show HTML Elements
- And many more ….

HOW TO WRITE JAVASCRIPT IN HTML

- The <script> Tag : In HTML, JavaScript code is inserted between <script> and </script> tags.

```
<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```

- You can place any number of scripts in HTML document, either in <head> or <body>
- Placing scripts at the bottom of the <body> element improves the display speed, because script interpretation slows down the display.
- External javascript: External scripts are practical when the same code is used in many different web pages. JavaScript files have the file extension **.js**.

```
<script src="myScript.js"></script>
```

CANVAS – THE PLAYGROUND

- HTML5 features the <canvas> element that allows you to draw 2D graphics using JavaScript.
- The <canvas> element requires at least two attributes: width and height that specify the size of the canvas:

```
<canvas width="500" height="300" id="canvas"></canvas>
```

- Like other elements, you can access the width and height properties of the <canvas> element via its DOM properties:

```
const canvas = document.querySelector('#canvas');
const width = canvas.width;// 500
const height = canvas.height;// 300
```

CANVAS – THE PLAYGROUND

- The <canvas> element features the getContext() method that returns a render context object.

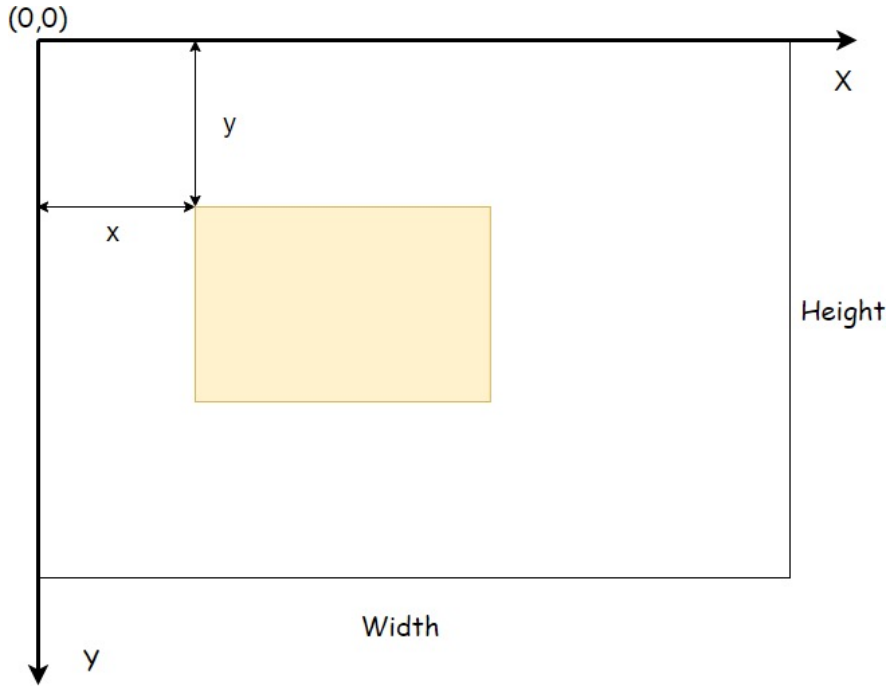
```
let canvas = document.querySelector('#canvas');
let ctx = main.getContext('2d');
```

- When using the <canvas> element, it's important to check if the browser supports the getContext() method. To do it, you use the following code:

```
let canvas = document.querySelector('#main');
if(canvas.getContext) {
   let ctx = main.getContext('2d');
}
```

THE 2D CONTEXT

- The 2D drawing context features methods for drawing simple 2D shapes such as paths, rectangles, and arcs.
- The coordinates in a 2D context begin at the upper-left of the <canvas> element, which is point (0,0) as shown in the picture on the right:



FILLS AND STROKES

Fill and stroke are two basic drawing operation on 2D drawing context.

- Fill fills in the shape with a specific style such as color, gradient, and image.
- Stroke adds colors to the edges of the shape.

The fillStyle and strokeStyle properties of the 2D drawing context will determine the fill and stroke styles.

By default, they both set to a value of '#000000'

```
(() => {
    const canvas = document.querySelector('#main');
    if (!canvas.getContext) {
        return;
    }

    // get the context
    let ctx = canvas.getContext('2d');

    // set fill and stroke styles
    ctx.fillStyle = '#F0DB4F';
    ctx.strokeStyle = 'red';

    // draw a rectangle with fill and stroke
    ctx.fillRect(50, 50, 150, 100);
    ctx.strokeRect(50, 50, 150, 100);

})();
```

JavaScript Canvas Demo

DRAWING WITH CANVAS

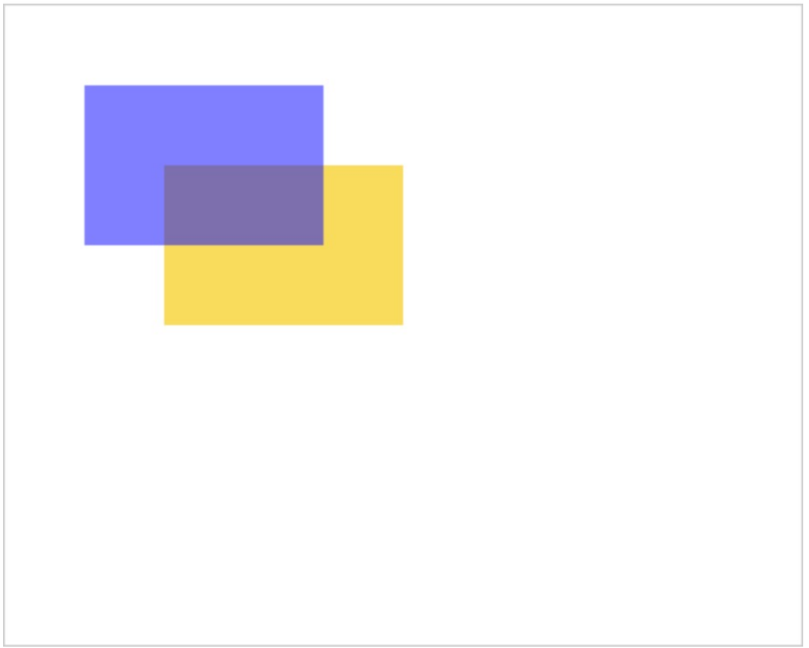
- The fillRect() is a method of the 2d drawing context object. The fillRect() method allows you to draw a filled rectangle at (x,y) position with a specified with and height on a canvas.

```
ctx.fillRect(x,y,width,height);
```

In this syntax:

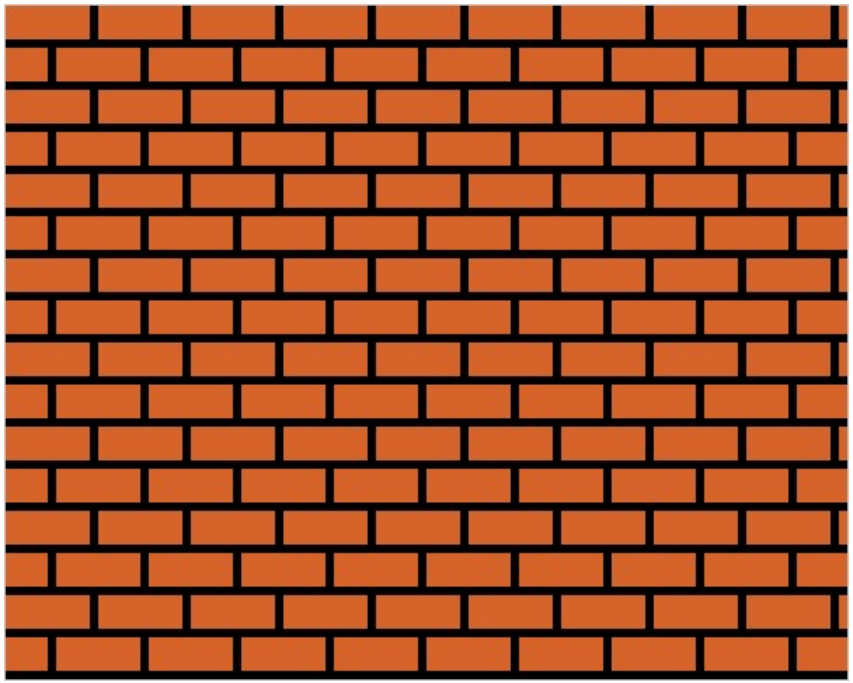
- x is the x-axis coordinate of the starting point of the rectangle.
- y is the y-axis coordinate of the starting point of the rectangle.
- width is the rectangles' width. It can be positive or negative. The positive values are to the right while the negative values are to the left.
- height is the rectangle's height. It can be also positive or negative. The positive values are down while the negative values are up.

```
function drawRectangles() {
    const canvas = document.querySelector('#canvas');

    if (!canvas.getContext) {
        return;
    }

    const ctx = canvas.getContext('2d');

    ctx.fillStyle = '#F9DC5C';
    ctx.fillRect(100, 100, 150, 100);

    ctx.fillStyle = 'rgba(0,0,255,0.5)';
    ctx.fillRect(200, 150, -150, -100);
}

drawRectangles();
```



PRACETICE JS 1:

- Draw a wall like this:



DATA TYPE

There are 8 basic data types in JavaScript.

- **number** for numbers of any kind: integer or floating-point, integers are limited by $\pm(2^{53}-1)$.
- **bigint** is for integer numbers of arbitrary length.
- **string** for strings. A string may have zero or more characters, there's no separate single-character type.
- **boolean** for true/false.
- **null** for unknown values – a standalone type that has a single value null.
- **undefined** for unassigned values – a standalone type that has a single value undefined.
- **object** for more complex data structures.
- **symbol** for unique identifiers.

Use **typeof()** to get data type of a variable

VARIABLE

- A variable is a “named storage” for data. We can use variables to store goodies, visitors, and other data.
- You can either use **let** or **var** to declare a variable

So what is the different between let and var?

NAMING & NAMING CONVENTION

```
var name = 'Robin Wieruch';

var Name = 'Dennis Wieruch';

var NAME = 'Thomas Wieruch';

console.log(name);
// "Robin Wieruch"

console.log(Name);
// "Dennis Wieruch"

console.log(NAME);
// "Thomas Wieruch"
```

Case Sensitive

```
// bad
var value = 'Robin';

// bad
var val = 'Robin';

// good
var firstName = 'Robin';
```

Meaningful

```
// bad
var firstname = 'Robin';

// bad
var first_name = 'Robin';

// bad
var FIRSTNAME = 'Robin';

// bad
var FIRST_NAME = 'Robin';

// good
var firstName = 'Robin';
```

camelCase recommended

NAMING & NAMING CONVENTION

```
// bad
var visible = true;

// good
var isVisible = true;

// bad
var equal = false;

// good
var areEqual = false;

// bad
var encryption = true;

// good
var hasEncryption = true;
```

How you should name boolean variable

```
// bad
function name(firstName, lastName) {
  return `${firstName} ${lastName}`;
}

// good
function getName(firstName, lastName) {
  return `${firstName} ${lastName}`;
}
```

How you should name function

NAMING & NAMING CONVENTION

```
class SoftwareDeveloper {
  constructor(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }
}

var me = new SoftwareDeveloper('Robin', 'Wieruch');
```

Use PascalCase to name class

WORK WITH STRING

You can write string with:

```
let single = 'single-quoted’;
let double = "double-quoted";
let backticks = `backticks`;



function sum(a, b) {
    return a + b;
}

alert(`1 + 2 = ${sum(1, 2)}.`); // 1 + 2 = 3.
```

WORK WITH STRING

Use backslash \ to mask special character

```
alert( 'I\'m the Walrus!' ); // I'm the Walrus!

alert( "\u00A9" ); // ©
alert( "\u{20331}" ); // 佱, a rare Chinese hieroglyph (long unicode)
alert( "\u{1F60D}" ); // 😍, a smiling face symbol (another long unicode)

let str1 = "Hello\nWorld"; // two lines using a "newline symbol"
```

Use + to concatenate strings

```
let s = "my" + "string";
alert(s); // mystring
```

WORK WITH STRING

Use **length** property to get string length

```
alert( `My\n`.length ); // 3
```

To get a character at position pos, use square brackets [pos] or call the method str.charAt(pos). The first character starts from the zero position:

```
let str = `Hello`; // the first character
alert( str[0] ); // H
```

Change the case: toLowerCase() and toUpperCase()

```
alert( 'Interface'.toUpperCase() ); // INTERFACE
alert( 'Interface'.toLowerCase() ); // interface
```

WORK WITH STRING

Searching for substring with str.indexOf(substr, pos).

```
let str = 'Widget with id’;
alert( str.indexOf('Widget') ); // 0, because 'Widget' is found at the beginning
alert( str.indexOf('widget') ); // -1, not found, the search is case-sensitive
alert( str.indexOf("id") ); // 1, "id" is found at the position 1 (..idget with id)
```

Get substring with str.slice(start [, end]) or str.substring(start [, end]) or str.substr(start [, length])

```
let str = "stringify";
alert( str.slice(0, 5) ); // 'strin', the substring from 0 to 5 (not including 5)
alert( str.slice(0, 1) ); // 's', from 0 to 1, but not including 1, so only
character at 0
```

WORK WITH NUMBER

The following math operations are supported:

- Addition +,
- Subtraction -,
- Multiplication *,
- Division /,
- Remainder %,
- Exponentiation **.

```
alert( 5 % 2 ); // 1, a remainder of 5 divided by 2
alert( 8 % 3 ); // 2, a remainder of 8 divided by 3

alert( 2 ** 2 ); // 4 (2 multiplied by itself 2 times)
alert( 2 ** 3 ); // 8 (2 * 2 * 2, 3 times)
alert( 2 ** 4 ); // 16 (2 * 2 * 2 * 2, 4 times)
```

FUNCTIONS

Functions are the main “building blocks” of the program. They allow the code to be called many times without repetition.

```
function showMessage() {
    alert( 'Hello everyone!' );
}
```

```
function SetName(parameter, otherParam) {
    ...body...
}
```

You can set default value to a parameter

```
function showMessage(from, text = "no text given") {
    alert( from + ": " + text );
}

showMessage("Ann"); // Ann: no text given
```

FUNCTIONS

A function can return a value back into the calling code as the result.

```
function sum(a, b) {
   return a + b;
}

let result = sum(1, 2);
alert( result ); // 3
```

CONTROL STATEMENT

The “If” statement

```
let year = prompt('In which year was ECMAScript-2015 specification published?', '');
if (year == 2015) alert( 'You are right!' );
```

A number 0, an empty string "", null, undefined, and NaN all become false. Because of that they are called “falsy” values.

The “If/else/else if” statement

```
let year = prompt('In which year was the ECMA2015 specification published?', ’’);
if (year < 2015) {
   alert( 'Too early...' );
}
else if (year > 2015) { alert( 'Too late' ); }
else { alert( 'Exactly!' ); }
```

CONTROL STATEMENT

The conditional operator “?”

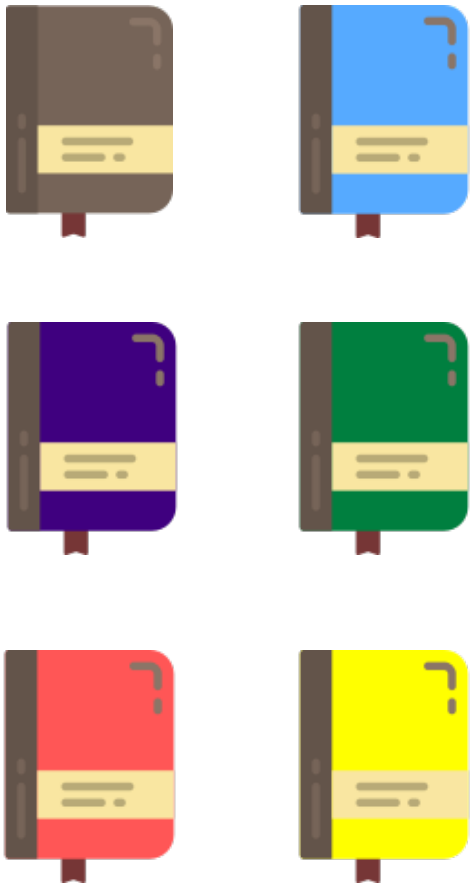
```
let result = condition ? value1 : value2;
```

So you have learnt quite a few things about Javascript

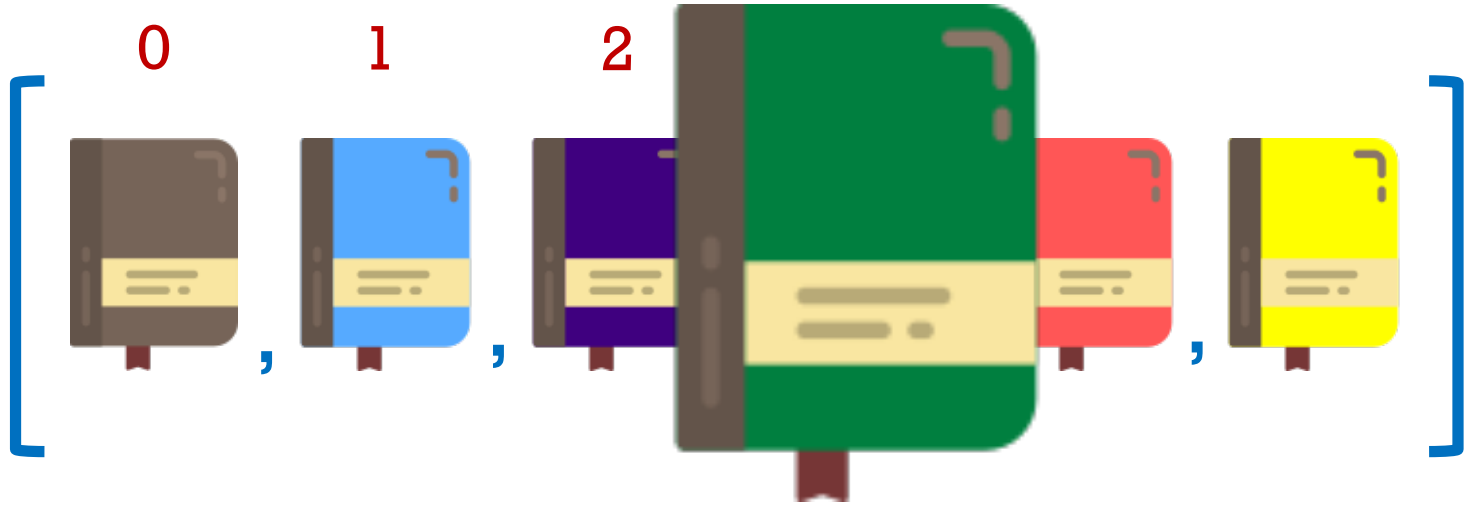
1. Javascript data type
2. How to declare variable : var & let
3. Some basic features with String and Number
4. Naming convention: How should you name variable, function and class
5. Control statement: If/Else/Else If
6. Onclick() event
7. The DOM Tree:
   - getElementById(),
   - getElementsByClassName()
   - Element.innerText
   - Element.innerHTML

What is an Array?

var book;

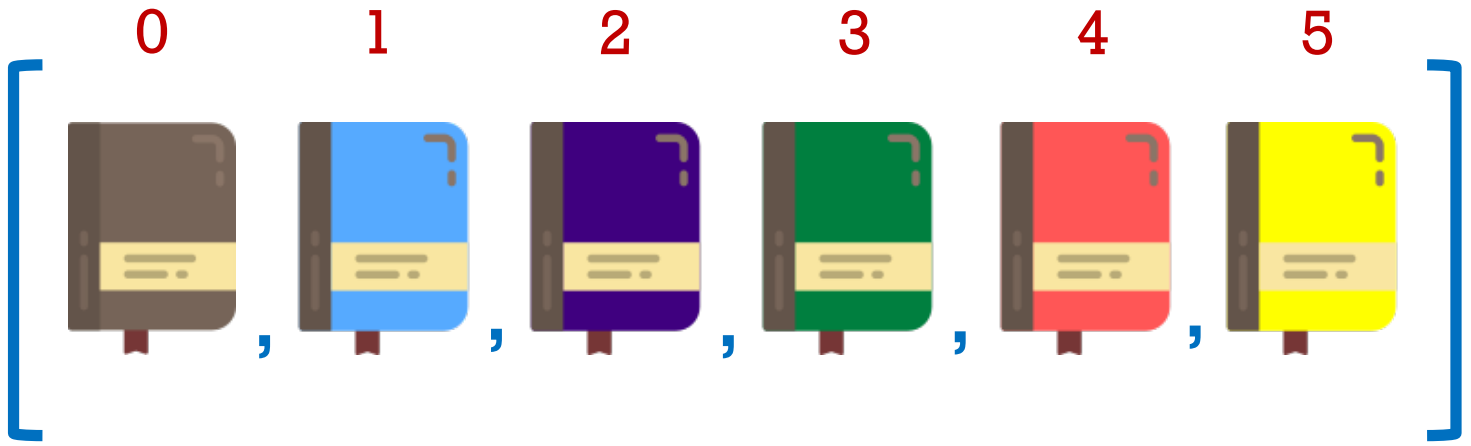


var books = new Array();
var books = []

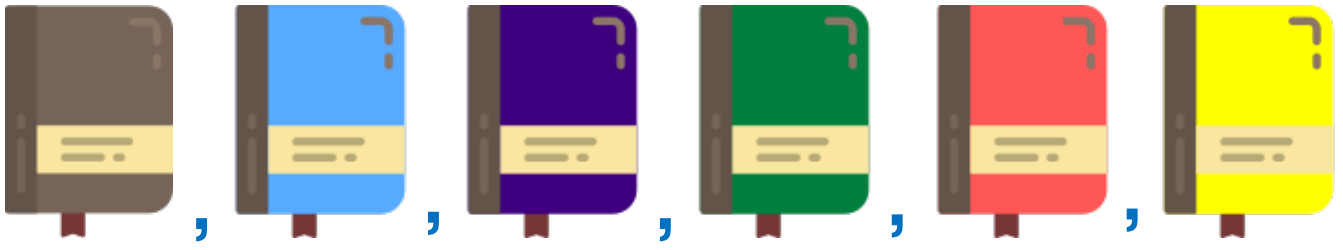
var books = 

var myBook = books[3];

var books =

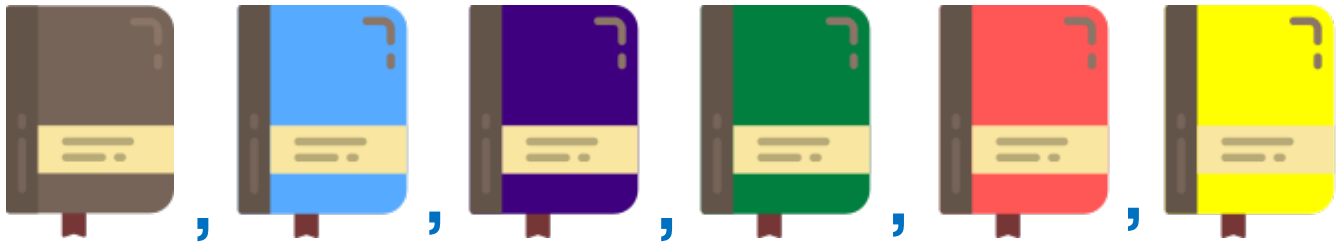



```
var myBook = books[3];
books.length;
```

var books = []

books.includes(); → true

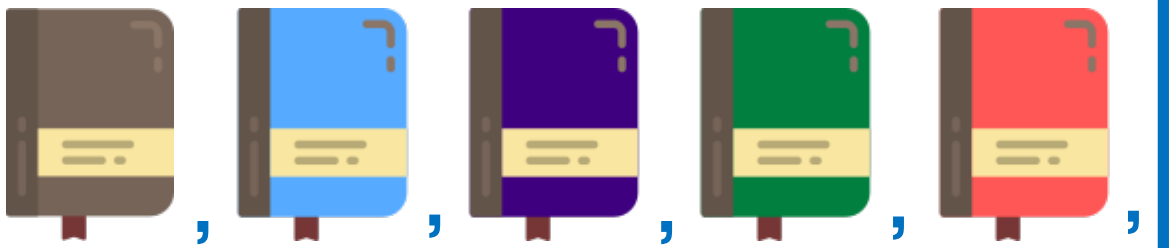
How to add/remove element to/from array?

var books = []

books.pop();

books.push();

How to add/remove element to/from array?

var books = []

books.pop();

books.push();

Similarly, you can also remove/add the first element of array

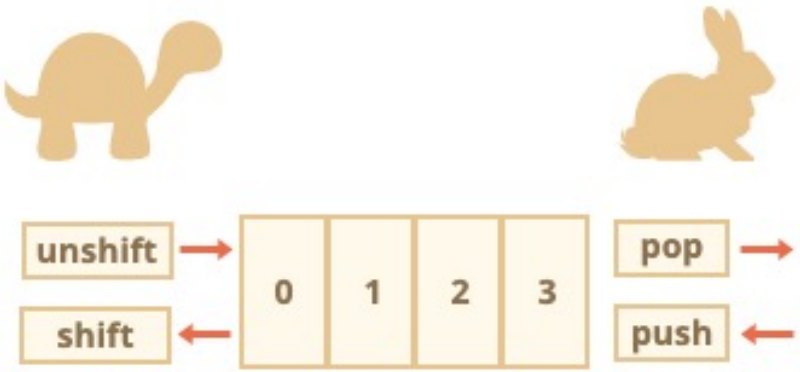
Remove the 1st element:

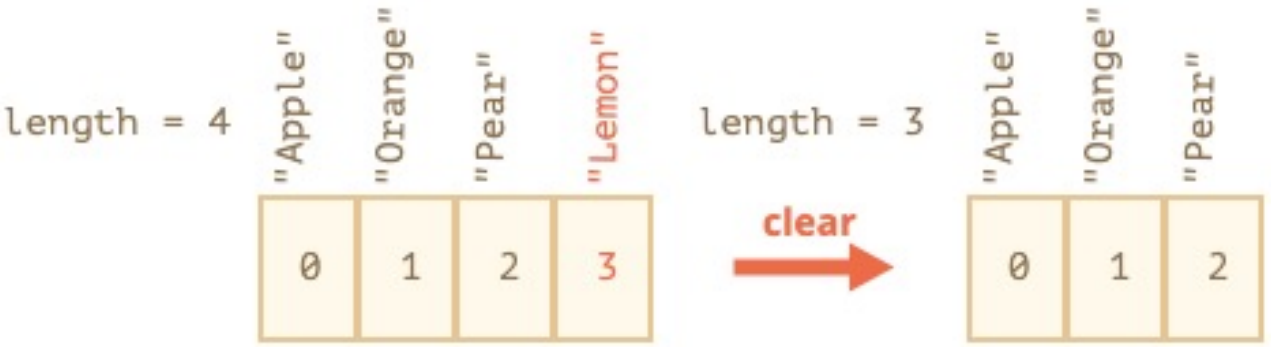
```
let fruits = ["Apple", "Orange", "Pear"];
alert( fruits.shift() ); // remove Apple and alert it
alert( fruits ); // Orange, Pear
```

Add to the 1st element:

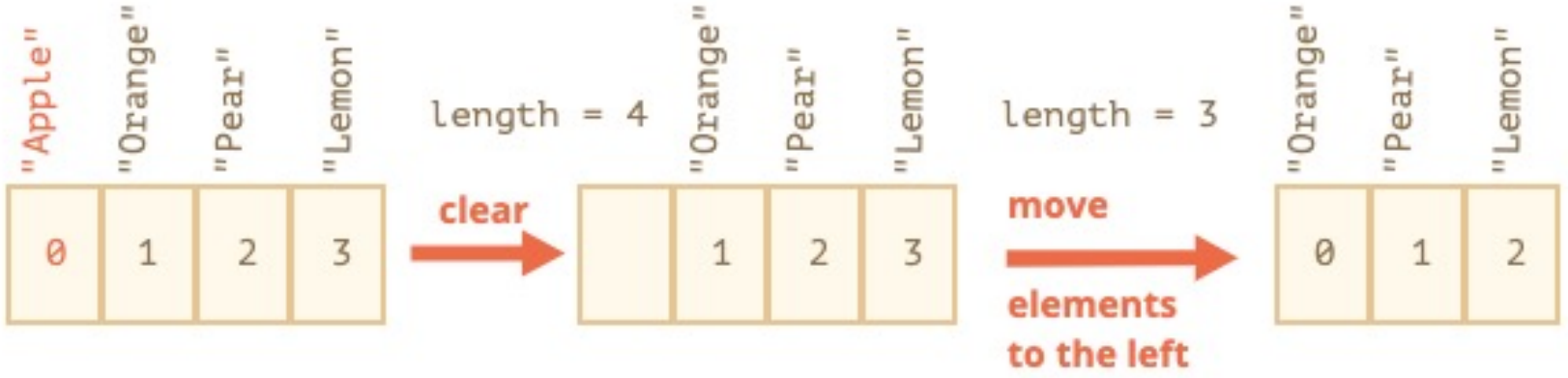
```
let fruits = ["Orange", "Pear"];
fruits.unshift('Apple’);
alert( fruits ); // Apple, Orange, Pear
```

Be aware that shift() and unshift() are quite slow compare to push() and pop()




length = 4
"Apple"
"Orange"
"Pear"
"Lemon"
0
1
2
3
clear
length = 3
"Apple"
"Orange"
"Pear"
0
1
2

pop()


"Apple"
"Orange"
"Pear"
"Lemon"
0
1
2
3
clear
length = 4
"Orange"
"Pear"
"Lemon"
1
2
3
move
elements
to the left
length = 3
"Orange"
"Pear"
"Lemon"
0
1
2

unshift()

Loop with array

```
let arr = ["Apple", "Orange", "Pear"];

for (let i = 0; i < arr.length; i++) {
      alert( arr[i] );
}
```

With a more elegant way:

```
let fruits = ["Apple", "Orange", "Plum"];

// iterates over array elements
for (let fruit of fruits) {
      alert( fruit );
}
```

You can also use:

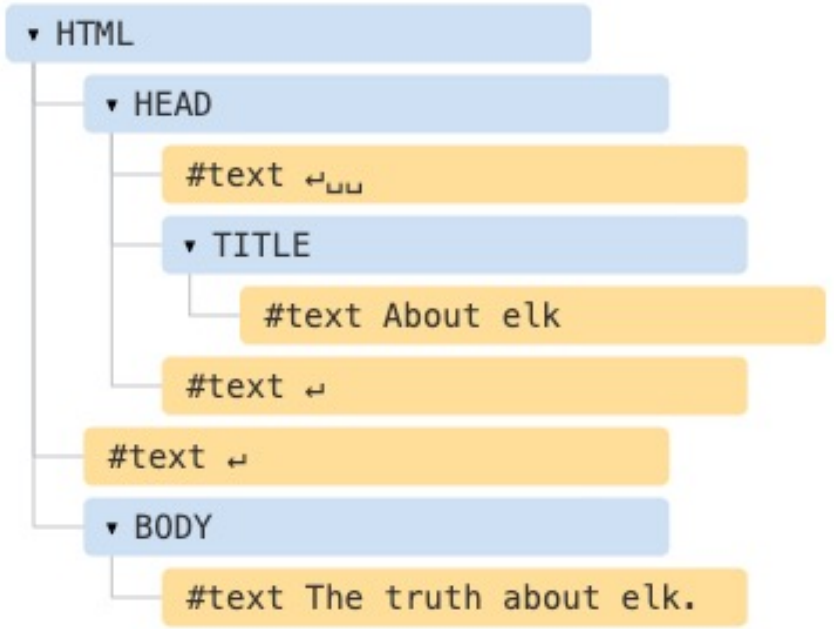
```
let arr = ["Apple", "Orange", "Pear"];

for (let key in arr) {
      alert( arr[key] ); // Apple, Orange, Pear
}
```

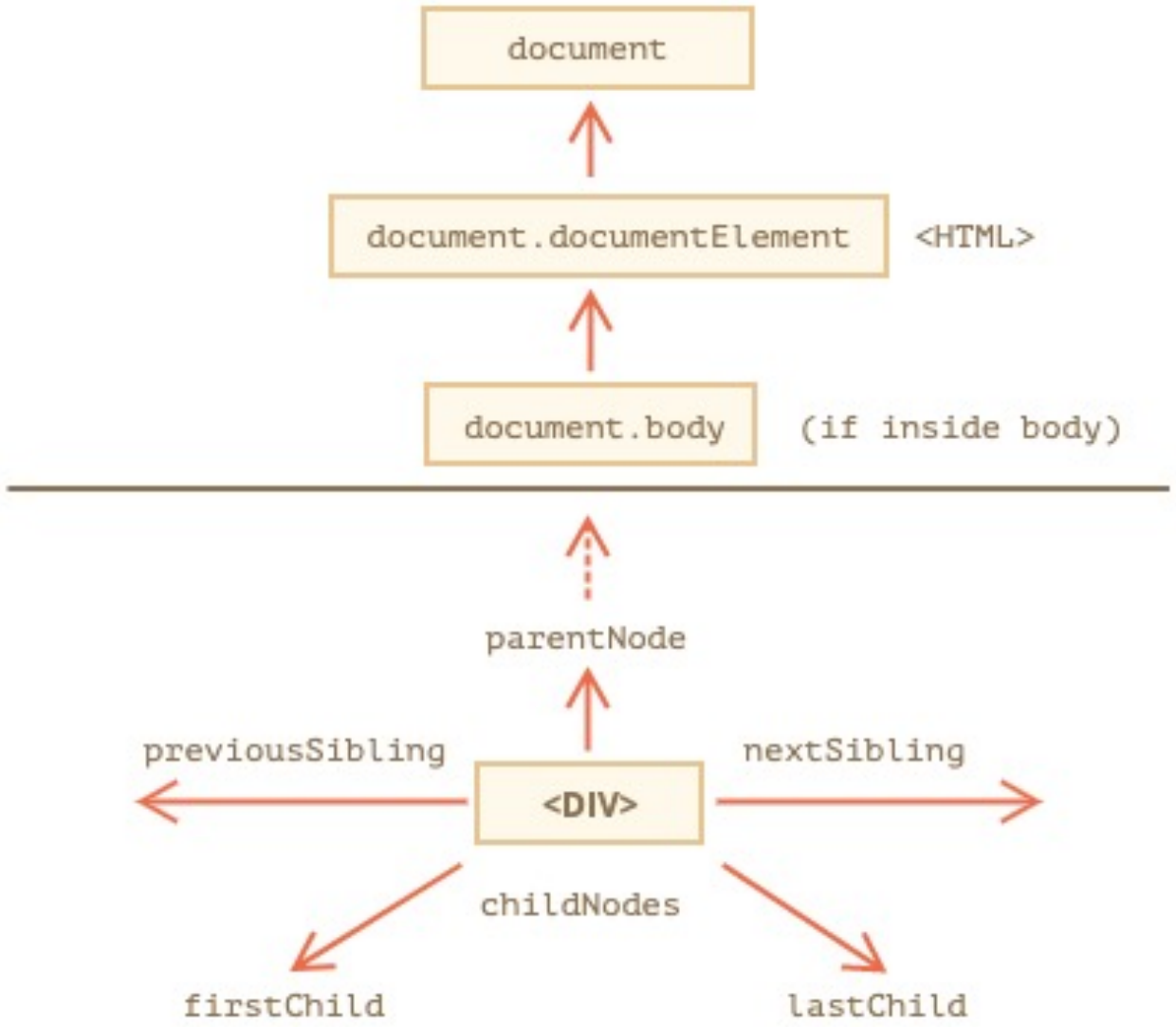
But it is not recommended!

What is DOM?

Document Object Model (DOM) represent all the element inside HTML.

```
1  <!DOCTYPE HTML>
2  <html>
3  <head>
4    <title>About elk</title>
5  </head>
6  <body>
7    The truth about elk.
8  </body>
9  </html>
```





<html> = document.documentElement

<body> = document.body

<head> = document.head

Child nodes (or children) – elements that are direct children. In other words, they are nested exactly in the given one.

Descendants – all elements that are nested in the given one, including children, their children and so on.

What is Browser Event?

Mouse events:

- click – when the mouse clicks on an element (touchscreen devices generate it on a tap).
- contextmenu – when the mouse right-clicks on an element.
- mouseover / mouseout – when the mouse cursor comes over / leaves an element.
- mousedown / mouseup – when the mouse button is pressed / released over an element.
- mousemove – when the mouse is moved.

Keyboard events:

- keydown and keyup – when a keyboard key is pressed and released.

Form element events:

- submit – when the visitor submits a <form>.
- focus – when the visitor focuses on an element, e.g. on an <input>.