Front-end development

Lesson 3 - Javascript Intro

adapt

What is Javascript?

Dynamic typing, interpreted, lightweight programming language mostly used for client side web development. Based on EcmaScript standard.

Hello world example in Javascript which prints out
'Hello world' on our document's <h1> tag:

```
let documentHeading = document.querySelector('h1');
documentHeading.textContent = "Hello world!";
```

You can try it in your browser console on any page with <h1> tag and see what happens. Your first DOM manipulation in a browser!

Ways to execute Javascript in browser

- Direct inline method adding **<script>** tag directly inside HTML document code:

```
<script>
  let documentHeading = document.querySelector('h1');
  documentHeading.textContent = "Hello world!";
</script>
```

- Adding **src** attribute to our <script> tag requesting javascript file named ‘helloWorld.js.’ from a relative directory:

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

Inline javascript execution example:

As you can see, **<script>** tag is placed directly inside HTML code:

```
<html>
  <head>
    <title>Inline javascript example</title>
  </head>
  <body>
    <h1>Inline javascript example heading</h1>

    ## Begin inline javaScript
    <script>
      let documentHeading = document.querySelector('h1');
      documentHeading.textContent = "Hello world!";
    </script>
    ## End inline javaScript

  </body>
</html>
```

External javascript execution example

<script> tag here is placed inside HTML code <head> section and attribute **src** specifies location of our *helloWorld.js* file:

```
<html>
  <head>
    <title>External javascript example</title>

    ## Begin external javaScript
    <script src="./helloWorld.js"></script>
    ## End external javaScript

  </head>
  <body>
    <h1>External javascript example heading</h1>
  </body>
</html>
```

External *helloWorld.js* file:

```
let documentHeading = document.querySelector('h1');
documentHeading.textContent = "Hello world!";
```

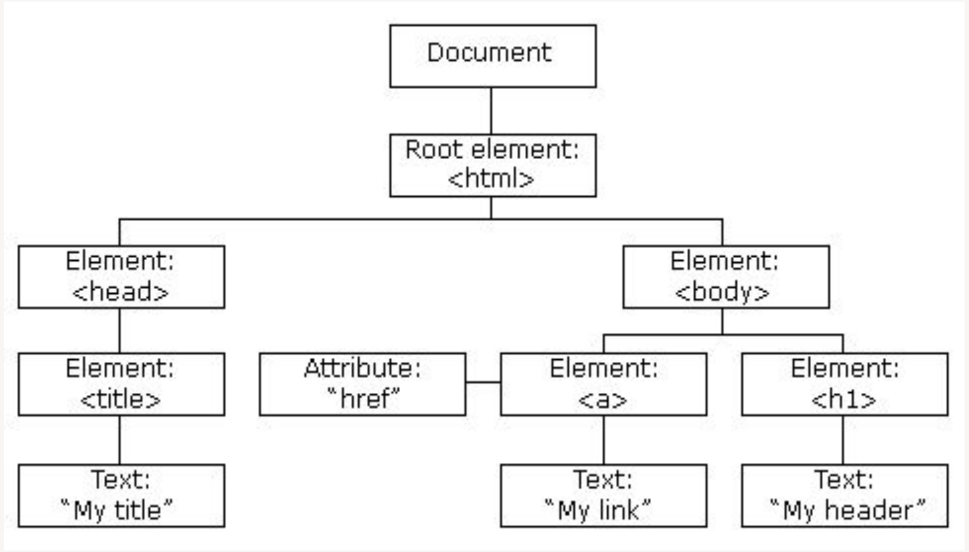
What is the expected result of this javascript code?

adapt

Working with DOM

Stands for ***Document Object Model***. It's the data representation of objects used to build structure and the content of the web page. Therefore, it could be accessed via javascript using DOM API. Styles, content and structure could be changed and manipulated:

DOM example:



adapt

Common DOM API methods

```
document.getElementById('hello');
document.getElementsByTagName('p');
document.querySelector('.hello');
document.querySelectorAll('#hello')
document.createElement('div');
element.appendChild(elementChild);
element.innerHTML = '<p>Hello world!</p>';
element.style.left = '20px';
element.setAttribute('data-name', 'John');
element.getAttribute('data-name');
element.addEventListener('click', (e) => alert('clicked!'));
```

adapt

Variables

JS variables are containers for storing data values for further access in your application.

```
const x = 5;
const y = 4;
const z = x + y // 9
```

Keywords for declaring variables:

- **var** - function scope variable. Can be global if defined outside function.
- **let** - block scope variable. Can be reassigned.
- **const** - block scope read only variable. Cannot be reassigned.
- without keyword - makes variable global, reachable everywhere (not recommended).

In modern web applications we mostly use **let** and **const** keyword for variables.

Operators

Arithmetic:

- Addition +
  - 2 + 2 = 4
- Subtraction: **-**
  - 4 - 2 = 2
- Multiplication: *****
  - 2 * 2 = 4
- Division: **/**
  - 4 / 2 = 2
- Modulus: **%**
  - 5 % 2 = 1
- Increment: **++**
  - x = 5; y = ++x; // 6
- Decrement: **--**
  - x = 5; y = --x; // 4

Comparison:

- Equal **==**
  - 5 == “5” // true
- Not equal: **!=**
  - 5 != 4 // true
- Strict equal: **===**
  - 5 === “5” // false
- Strict not equal: **!==**
  - 5 !== “5” // true
- Strict Equal: **===**
  - 5 === “5” // false
- Greater than: **>**
  - 5 > 5 // false
- Greater than or equal: >=
  - 5 >= 5 // true

Assignment:

- Variable assignment =
  - x = 5 // 5
- Addition assignment: **+=**
  - x += 5 // 10
- Subtraction assignment: **-=**
  - x -= 4 // 6

Logical:

- Logical and: **&&**
  - true && true // true
- Logical or: ||
  - true || false // true
- Logical not: **!**
  - !true // false

adapt

Data types and structures

Primitive types:

- **Boolean** - logical data type contains two values (**true** or **false**).
- **Number** - floating point or simple number.
- **BigInt** - greater precision number.
- **String** - characters representing textual data.
- **Null** - empty or non-existent value.
- **Undefined** - declared variable which has not been defined.

Reference types:

- **Object** - structures consisting of key-value pairs to describe data.
- Function, Array - but these are technically objects too.

adapt

Program control flow

If - else statement:

```
const x = 5;
const y = 6;

if (x > y) {
  console.log("x is greater than y.");
} else {
  console.log("x is not greater than y.");
}
// Result: x is not greater than y.
```

Switch statement:

```
const vegetableType = 'Cucumber';

switch (vegetableType) {
 case 'Cucumber':
   console.log('Cucumbers are $0.59 a pound.');
   break;
 case 'Spinach':
   console.log('Spinachs are $0.32 a pound.');
   break;
 case 'Tomato':
   console.log('Tomatoes are $0.48 a pound.');
   break;
 case 'Cabbage':
   console.log('Cabbages are $3.00 a pound.');
   break;
 default:
  console.log('Sorry, we are out of ' + vegetableType + '.');
}
// Result: Cucumbers are $0.59 a pound.
```

adapt

Falsy values in Javascript

when a result of an expression or a value of a variable evaluates to one of the following values, this expression will become false when used inside **if … else** statement:

- false
- undefined
- null
- 0
- NaN
- Empty string “”

Objects

Objects are used to describe a real life object by combining its properties and methods:

- **Properties** - define characteristics of object.
- **Methods -** define operations with current object.

```
const myHonda = {
  color: 'red',
  wheels: 4,
  engine: {
    cylinders: 4,
    size: 2.2
  },
  getColor: function() { return this.color; }
};

console.log(myHonda.getColor());
// Result: red
```

- Object literal is defined using curly brackets **{}**
- Two ways to access object properties:
  - myHonda.color
  - myHonda[‘color’]

- Add new property to an object:
  - myHonda.frameType = ‘Coupe’

- Requesting non existing property returns *undefined*:
  - myHonda.seatCount // undefined

Arrays

- Array stores a data list to a single variable. You can create empty array with a straight braces: []
- Arrays are list-like objects. You can perform traversal operations against them to add, filter or modify it’s data.
- Strings cannot be used as array indexes. Instead each element of array is indexed by numbers. Indexes do count from 0. First element has index 0, second - 1, third - 2 etc.

```
const numberList = [1, 2, 3, 4, 5];
const letterList = ['H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd'];

console.log(numberList.length) // Result: 5.
console.log(letterList.length) // Result: 11.
console.log(numberList[1]) // Result 2.
numberList.push(6) // Adds element to array.
console.log(numberList) // Result: [1, 2, 3, 4, 5, 6].
numberList.pop() // Removes last item from array.
console.log(numberList) // Result: [1, 2, 3, 4, 5].
console.log(letterList.join('')); // Result: Hello world
```

Loops and iterations

Loops are common way to do something repeatedly or iterate through a data list. There are different statements used to create a loop, but essentially they do same thing - perform some repeated routine: Commonly used statements:

- **for** - iterates till value of index meets the condition: ***for (let i = 0; i < catList.length; i++) { ...operations with i (index) }***
- **for ... in** - iterates through object providing key in each iteration: ***for (let key in catList) { ... operations with key }***
- **while** - iterates while specified condition evaluates to true: ***while (counter < 5) { ...do something while counter < 5 }***
- **for ... of** - Iterates an array providing current value in each iteration: ***for (let value of catList) { ...do something with value }***

For loop:

```
var catList = ['Meow', 'Siuzi', 'Tracy'];

for (let i = 0; i < catList.length; i++) {
  console.log(catList[i]);
  // 'Meow', 'Siuzi', 'Tracy'.
}
```

For … in loop:

```
var catList = ['Meow', 'Siuzi', 'Tracy'];

for (let key in catList) {
  console.log(key);
  // Logs: 0, 1, 2, 3, 4.
}
```

Functions

Functions are fundamental building blocks in Javascript. They are a set of procedures/logic to perform a certain task or calculate and return a value. Function definition consist of *function* keyword followed by a name of function, list of parameters enclosed in **()** separated by commas and function body consisting of javascript statements is enclosed in curly brackets **{}**.

```
// Returns the value of two numbers.
function getSum(a, b) {
  return a + b;
}

// Functions can be assigned to a variable.
const multipleItems = function(a, b) {
  return a * b;
};

console.log(getSum(2, 2)) // Result: 4.
console.log(multipleItems(2, 4)) // Result: 8.
```

Json

Stands for **Javascript object notation**. Is a lightweight data interchange format. Easy for humans to read. This format is common way to be served through APIs.

```json
{
 "firstName": "John",
 "lastName": "Smith",
 "isAlive": true,
 "age": 27,
 "address": {
   "streetAddress": "21 2nd Street",
   "city": "New York",
   "state": "NY",
   "postalCode": "10021-3100"
 },
 "phoneNumbers": [
   {
     "type": "home",
     "number": "212 555-1234"
   },
   {
     "type": "office",
     "number": "646 555-4567"
   },
   {
     "type": "mobile",
     "number": "123 456-7890"
   }
 ],
 "children": [],
 "spouse": null
}
```

Fetching data from API.

API stands for *Application programming interface.* Majority of work building web applications consist of interacting with APIs in some way or another. Either you make a http **POST** request or **GET**.

Example API link: https://cat-fact.herokuapp.com/facts/random?amount=5

There are two build in native methods of making API calls in javascript:

- XMLHttpRequest (XHR).
- Fetch (Modern replacement for XHR)

XMLHttpRequest

Use XMLHttpRequest (XHR) objects to interact with servers. You can retrieve data from a URL without having to do a full page refresh. This enables a Web page to update just part of a page without disrupting what the user is doing. XMLHttpRequest is used heavily in AJAX programming.

```javascript
// AJAX request example using XMLHttpRequest.
function loadJson(path) {
  const xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState === 4 && this.status === 200) {
      const result = JSON.parse(this.responseText);
      console.log(result);
    }
  };
  xhttp.open("GET", path, true);
  xhttp.send();
}

loadJson('/examples/basics/12.%20jsonExample.json');
// Result: {firstName: "John", lastName: "Smith", isAlive: true, age: 27, address: {…}, …}
```

adapt

Fetch

The Fetch API provides an interface for fetching resources (including across the network). It will seem familiar to anyone who has used XMLHttpRequest, but the new API provides a more powerful and flexible feature set.

```javascript
// AJAX request example using promise based fetch method.
function loadLocalJson(path) {
  return fetch(path)
    .then(response => {
      if (!response.ok) {
        throw new Error('Network response was not ok.');
      }
      return response.json();
    })
    .then(result => console.log(result))
    .catch(err => console.log('Error message:', err));
}

loadLocalJson('/examples/basics/12.%20jsonExample.json');
// Result: {firstName: "John", lastName: "Smith", isAlive: true, age: 27, address: {…}, …}
```

Javascript frameworks

Vanilla JS term is often used to represent a plain Javascript without any additional framework/library. That’s what you should mostly aim for because frameworks come and go, but core principles do stay the same.

“A software framework is an abstraction in which software providing generic functionality can be selectively changed by additional user-written code.” - hackr.io

Most popular JS frameworks ones as in 2019:

React ANGULAR Vue.js

adapt