Internet Technologies

Introduction to JavaScript

University of Cyprus
Department of Computer Science

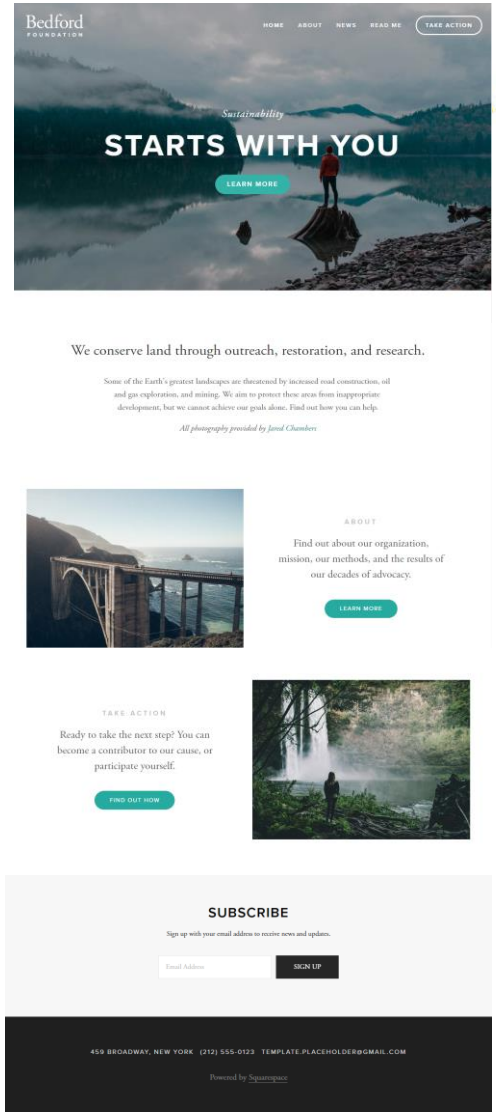
What we've learned so far

Describes the content and structure of the page

+

Describes the appearance and style of the page

produces



A web page… that doesn't do anything!!!

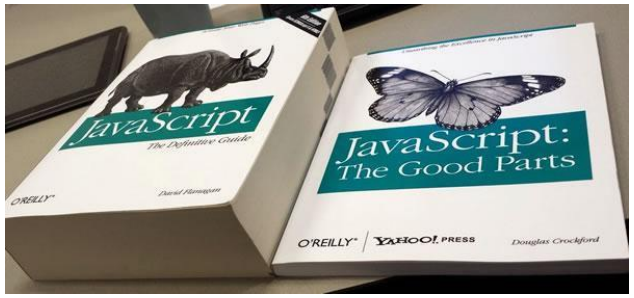
What we've learned so far

- We've learned how to build web pages that:
  - Look the way we want them to
  - Can link to other web pages
  - Display differently on different screen sizes (responsive)
- But we don't know how to build web pages that **do** something like:
  - Get user input
  - Save user input
  - Show and hide elements when the user interacts with the page
  - etc.

… so we enter JavaScript

JavaScript

- JavaScript is a programming language.
  - has nothing to do with Java; said to be named that way for marketing reasons
  - first version was written in 10 days
  - several fundamental language decisions were made because of company politics and not technical reasons



- However, it is currently the only programming language that your browser can execute natively.
- Therefore if you want to make your web pages do stuff, you must use JavaScript: There are no other options.

ECMAScript

- European Computer Manufacturers Association (ECMAScript) or (ES) is a standard for scripting languages like JavaScript, ActionScript and Jscript
  - JavaScript was invented by Brendan Eich in 1995, and became an ECMA standard in 1997
- ECMAScript was initially created to standardize JavaScript, which is the most popular implementation of ECMAScript and ensure the interoperability of web pages across different browsers
  - Early ECMAScript versions have been abbreviated to ES1, ES2, ES3, ES5, and ES6
  - Since 2016, versions are named by year (ECMAScript 2016, 2017, 2018, 2019, 2020, 2021, 2022) – see version history and updates per edition

Code in web pages

```
<!DOCTYPE html>
<html>
  <head>
   <title>CS 425</title>
   <link rel="stylesheet" href="style.css" />
   <script src="script.js"></script>
  </head>
  <body>
   ... contents of the page...
  </body>
</html>
```

console.log

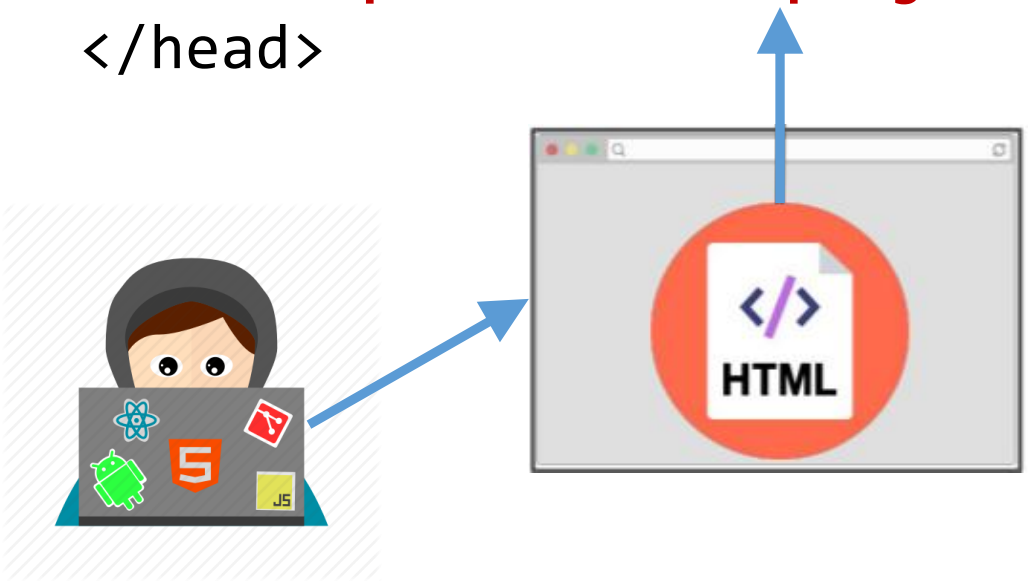
- You can print **log** messages in JavaScript by calling `console.log()`:

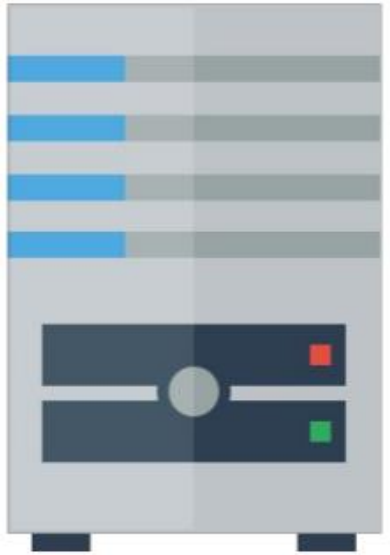
```JS
console.log('Hello, world!');
```

- This JavaScript's equivalent of Java's System.out.println, C’s printf, etc.

How does JavaScript get loaded?

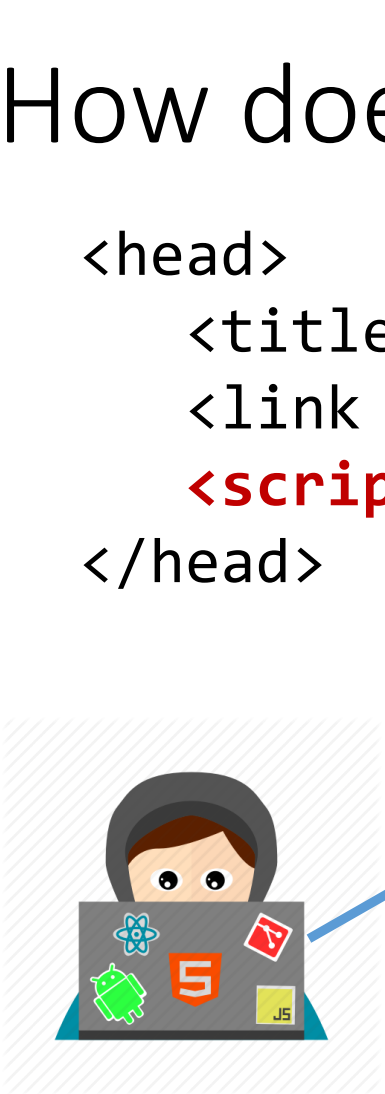
```
<head>
    <title>CS 425</title>
    <link rel="stylesheet" href="style.css" />
    <script src="script.js"></script>
</head>
```



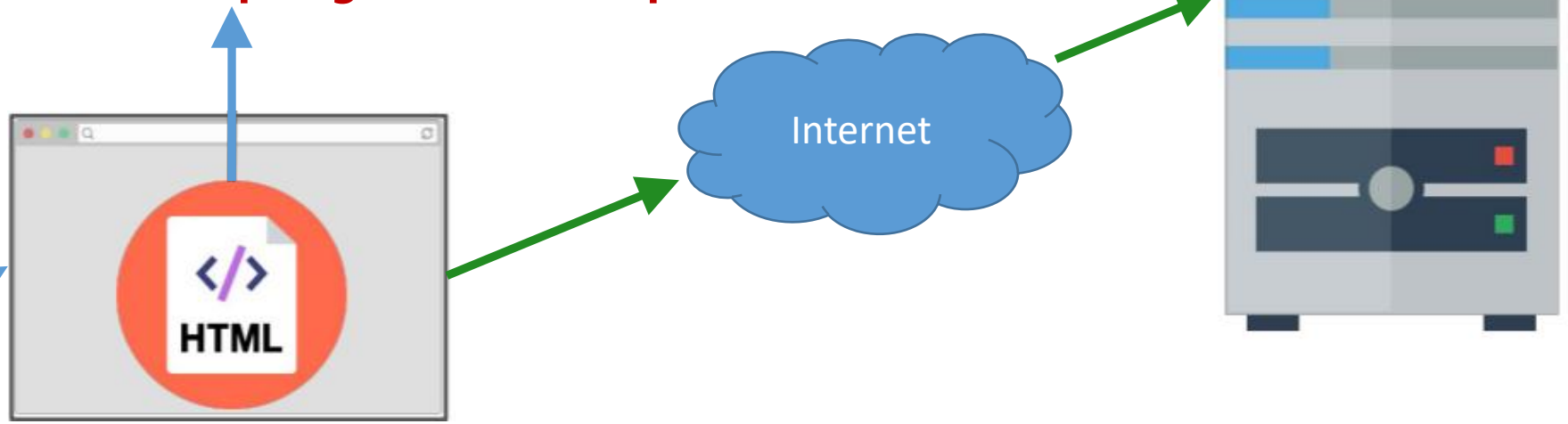


The browser is parsing the HTML file, and gets to a script tag, so it knows it needs to get the script file as well.

How does JavaScript get loaded?

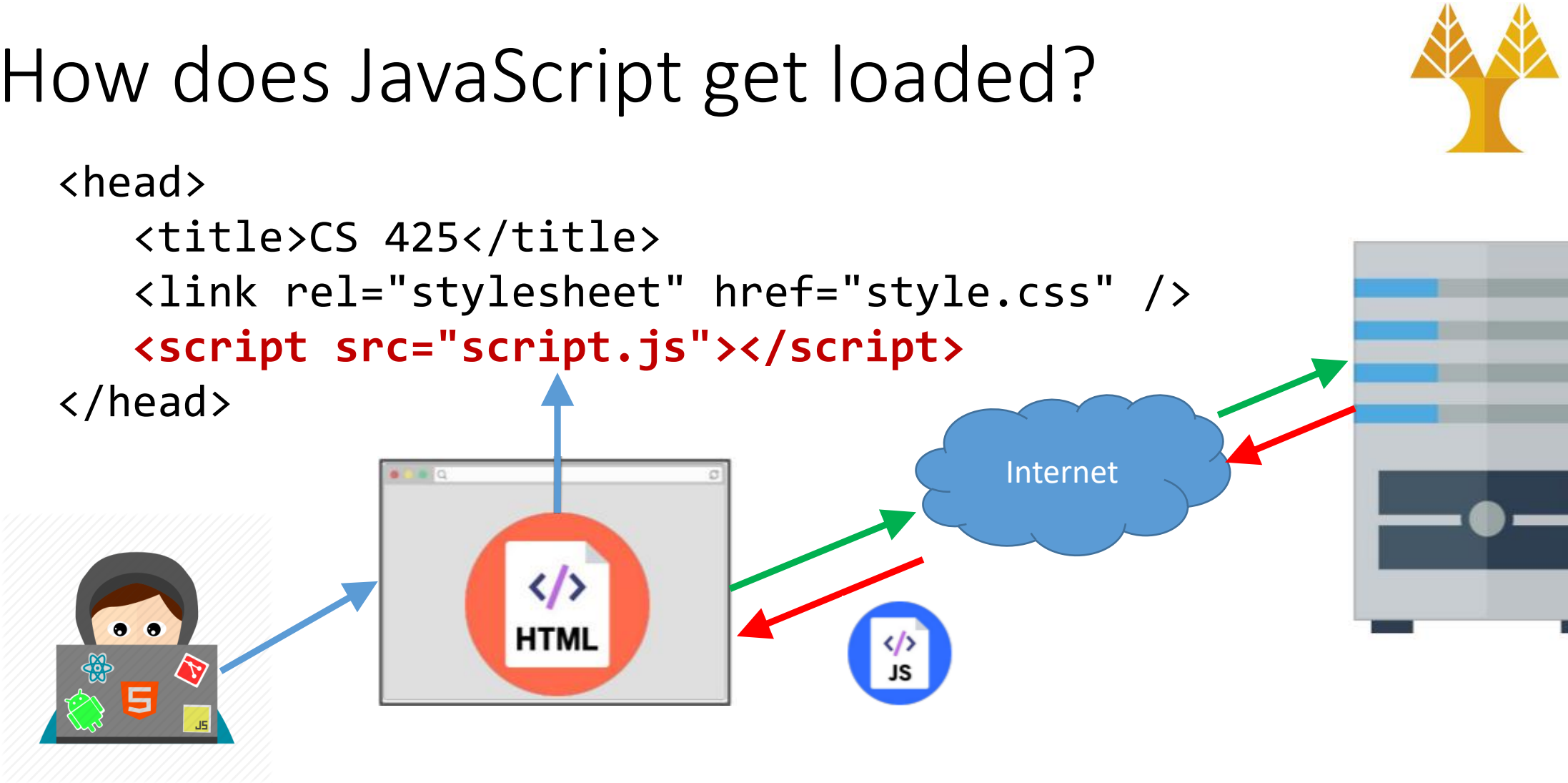


```
<head>
    <title>CS 425</title>
    <link rel="stylesheet" href="style.css" />
    <script src="script.js"></script>
</head>
```



The browser makes a request to the server for the script.js file, just like it would for a CSS file or an image...

How does JavaScript get loaded?

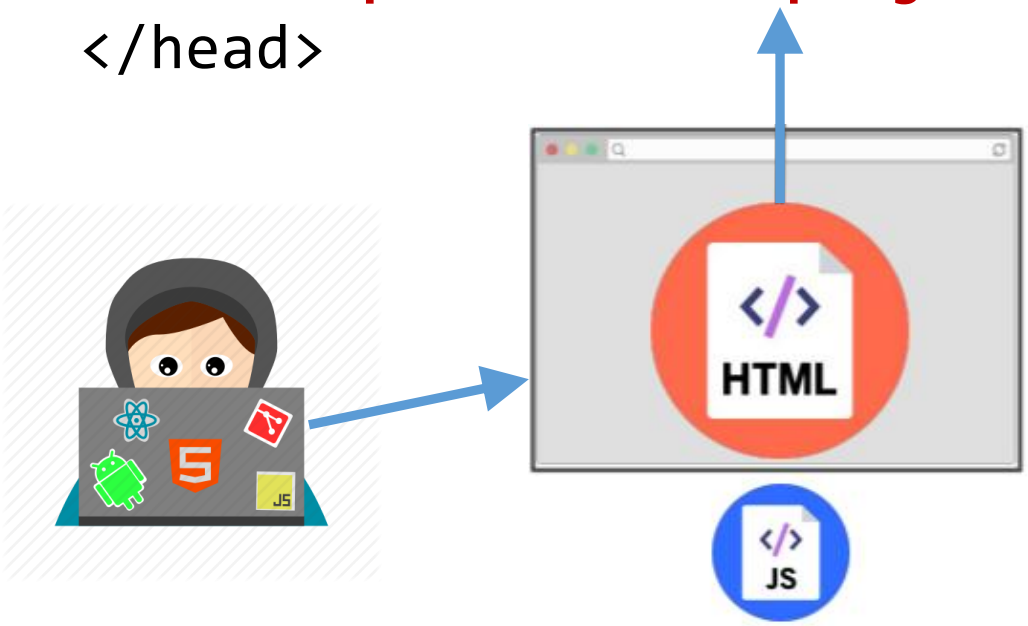


```
<head>
    <title>CS 425</title>
    <link rel="stylesheet" href="style.css" />
    <script src="script.js"></script>
</head>
```


And the server responds with the JavaScript file, just like it would with a CSS file or an image...

How does JavaScript get loaded?

```
<head>
    <title>CS 425</title>
    <link rel="stylesheet" href="style.css" />
    <script src="script.js"></script>
</head>
```




```
console.log('Hello, world!');
```
JS

Now at this point, the JavaScript file will **execute "client-side"**, or in the browser on the user's computer

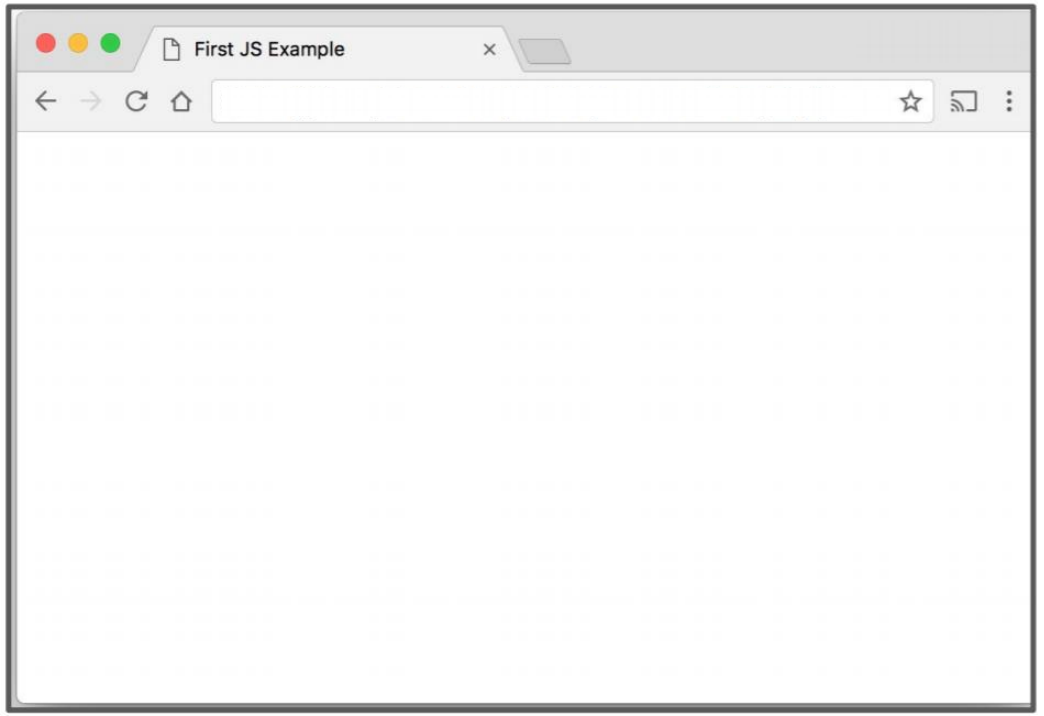
JavaScript execution

- There is **no "main method"**
  - The script file is executed from top to bottom.
- There's **no compilation** by the developer
  - JavaScript is compiled and executed on the fly by the browser

Simple Example

HTML

```
<!DOCTYPE html>
<html>
  <head>
   <title>First JS Example</title>
   <script src="script.js"></script>
  </head>
  <body>
</body>
</html>
```



Nothing happened!

script.js

```
console.log('Hello, world!');
```

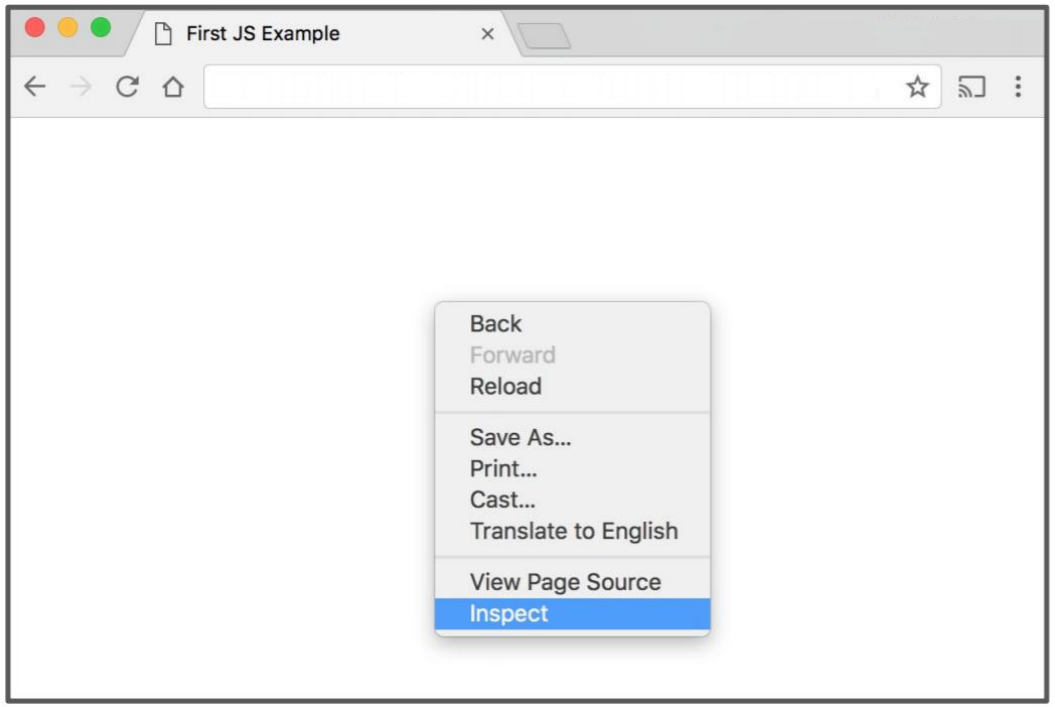
Simple Example

HTML

```
<!DOCTYPE html>
<html>
  <head>
   <title>First JS Example</title>
   <script src="script.js"></script>
  </head>
  <body>
</body>
</html>
```

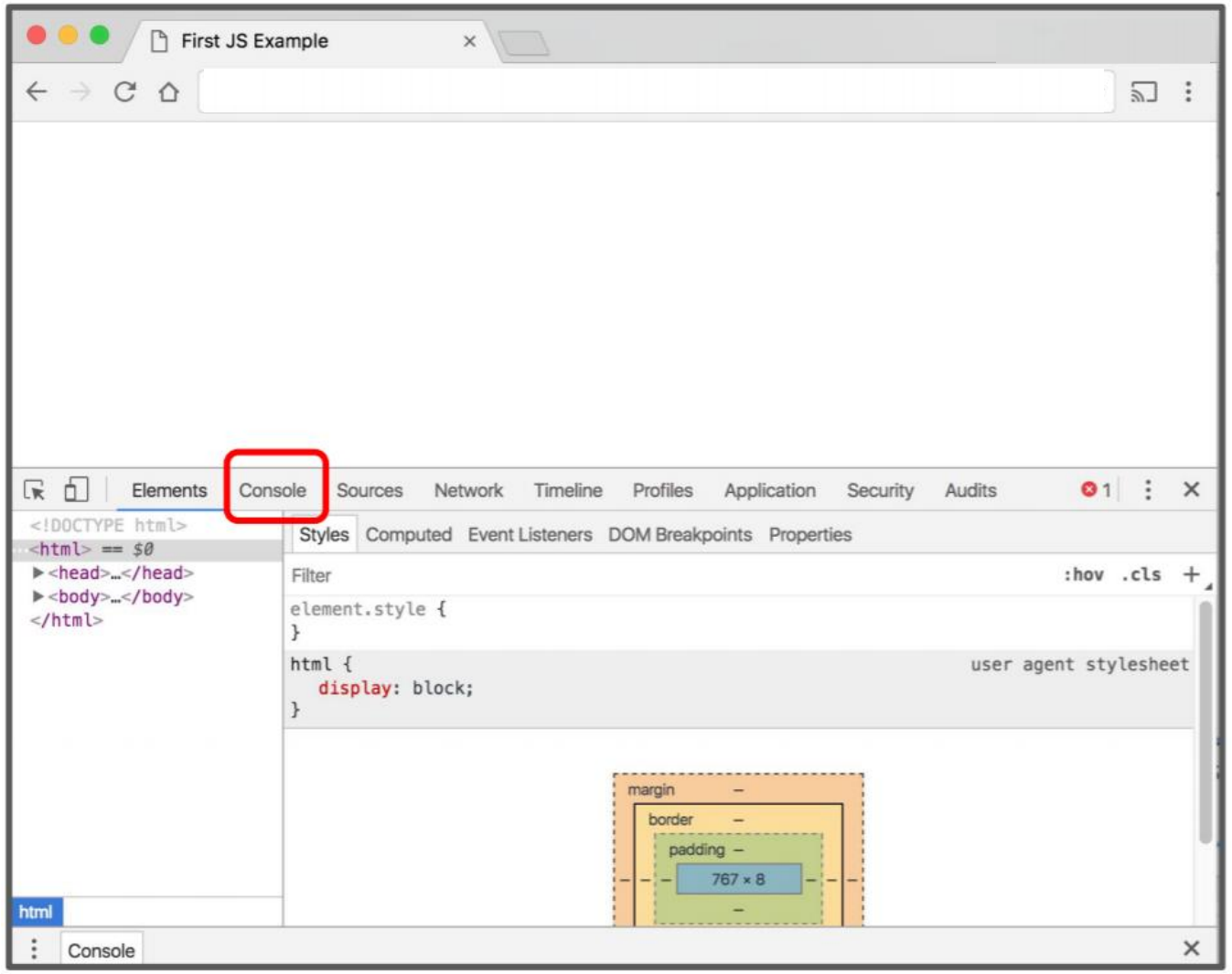
script.js

```
console.log('Hello, world!');
```

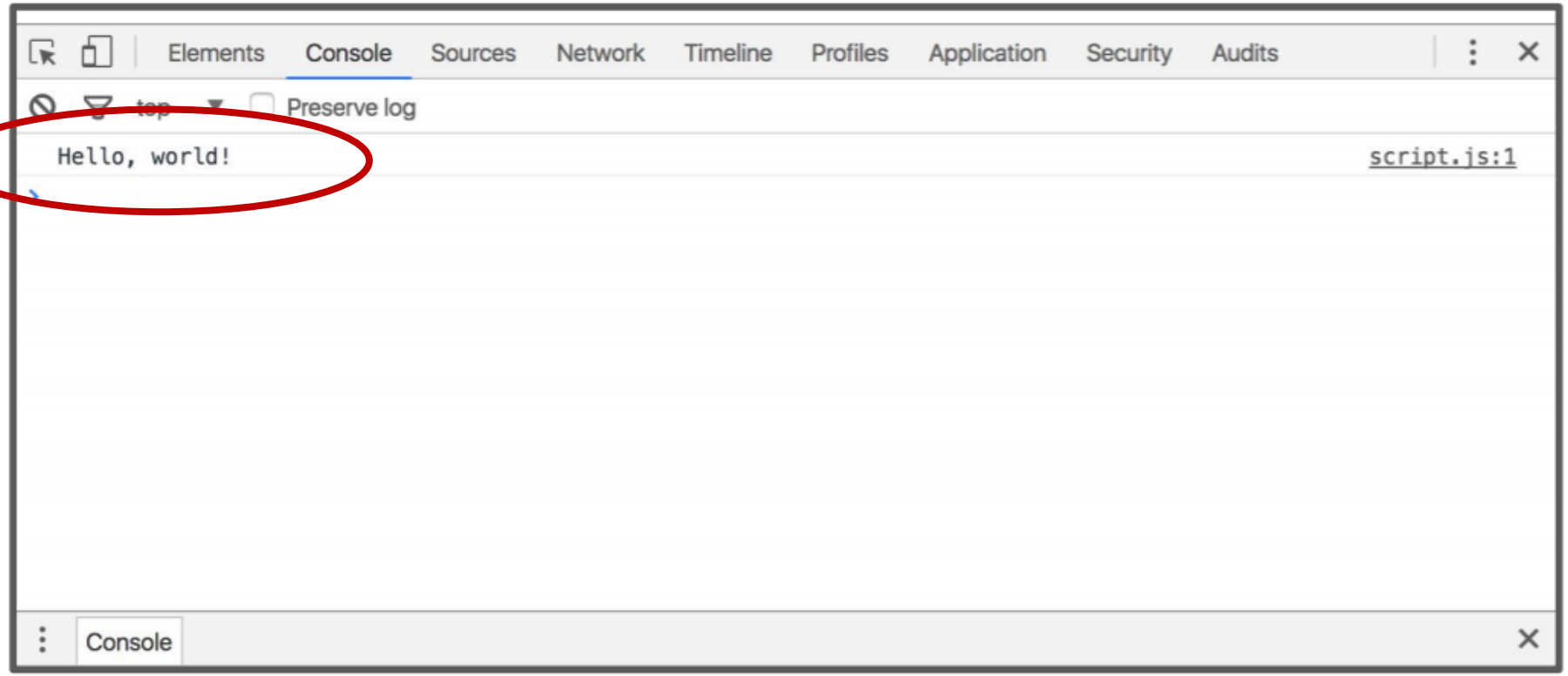


Right-click (or control-click on Mac) and choose "Inspect"

Click "Console" tab



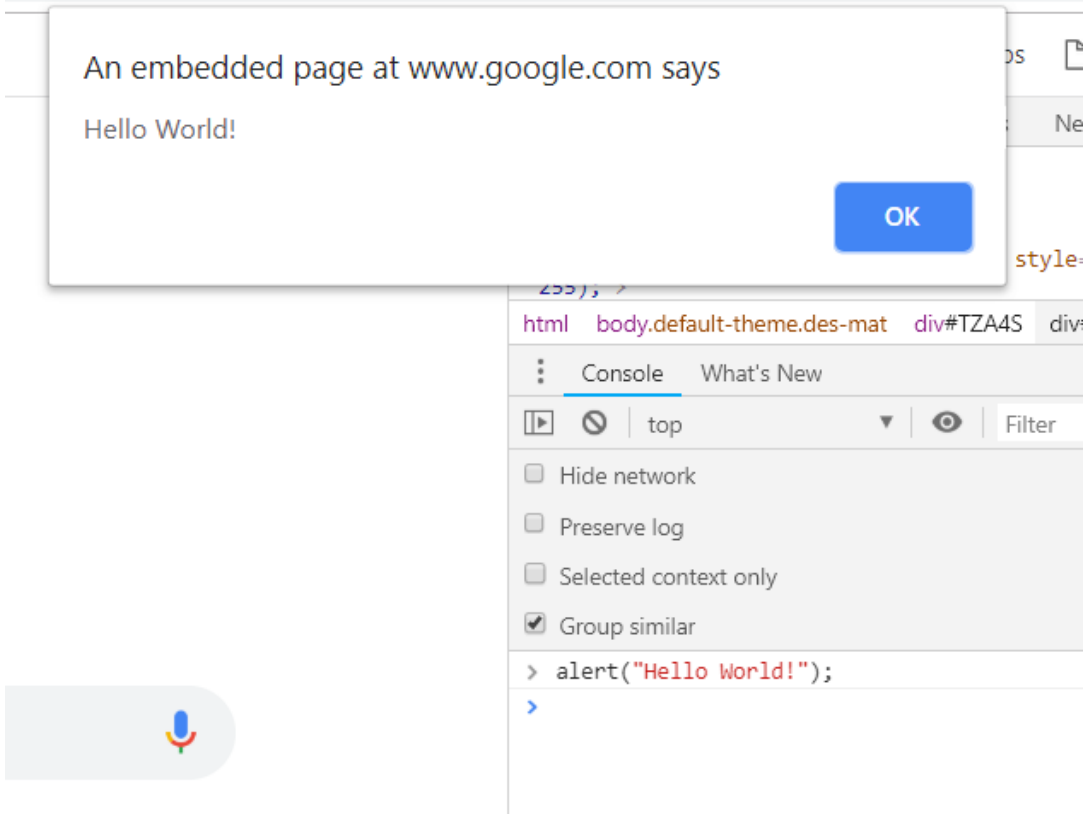
Console



The "Console" tab is also a (Read-Eval-Print Loop, REPL), or an interactive language shell, so you can type in JavaScript expressions, etc. to test out the language.

alert() method

- The alert() method displays an alert box with a specified message and an OK button.
- An alert box is often used if you want to make sure information comes through to the user.
- **Note:** The alert box takes the focus away from the current window and forces the browser to read the message. Do not overuse this method, as it prevents the user from accessing other parts of the page until the box is closed.



Same as Java/C++/C-style languages

- for-loops:

```
for (let i = 0; i < 5; i++) { ... }
```

- while-loops:

```
while (notFinished) { ... }
```

- comments:

```
// comment or /* comment */
```

- conditionals (if statements):

```
if (...) {
   ... }
else {
   ... }
```

Variables: `var`, `let`, `cons`

- Declare a variable in JS with one of three keywords:

```JS
// Function scope variable
var x = 15;
// Block scope variable {}
let fruit = 'banana';
// Block scope constant; cannot be reassigned
const isHungry = true;
```

- You do not have to declare the datatype of the variable before using it ("dynamically typed")

What's a "block"?

- In the context of programming languages, a block is a group of 0 or more statements, **usually surrounded by curly braces**. (wiki / mdn)
  - Also known as a compound statement
  - Not JavaScript-specific; exists in most languages (C++, Java, Python, etc)
  - Has absolutely nothing to do with the HTML/CSS notion of "block", i.e. block elements
- A block might look like

```
{                                                    JS
   console.log('Hello!');
   console.log('Welcome to JavaScript');
}
```

Variables best practices

- Use `const` whenever possible.
- If you need a variable to be reassignable, use `let`.
- Not doing so will result in global variables.
  - We want to avoid polluting the global namespace.
- **Avoid using var.**
  - You will see a ton of example code on the internet with `var` since `const` and `let` are relatively new.
  - However, `const` and `let` are well-supported, so there's no reason not to use them.
  - (This is also what the Google and AirBnB JavaScript Style Guides recommend.)

Aside: The internet has a ton of misinformation about JavaScript!

Including several "accepted" StackOverflow answers, tutorials, etc. Lots of stuff online is years out of date. Treat carefully.

Types

- JS **variables** do not have types, but the **values** do.
- There are six primitive types (mdn):
  - Boolean: true and false
  - Number: everything is a double (no integers)
  - String: in 'single' or "double-quotes"
  - undefined: **variable has been declared** but has not yet been assigned a value
  - null: is an assignment value, but null is a value meaning "this has no value"
- There are also Object types, including Array, Date, String (the object wrapper for the primitive type), etc.

Boolean

- There are two literal values for boolean:
  - `true` and `false` that behave as you would expect
- Can use the usual boolean operators: && || !
- Non-boolean values can be used in control statements, which get converted to their "truthy" or "falsy" value:
  - null, undefined, 0, NaN, '' , "" evaluate to `false`
  - Everything else evaluates to `true`

JS

```
if (username) {
   // username has been declared
   // and assigned a non null value
}
```

Numbers

- `const homework = 0.45;`
- **All numbers are floating point real numbers**.
- No integer type.
- Operators are like Java or C++.
- Precedence like Java or C++.
- A few special values: `NaN` (not-a-number), `+Infinity`, `-Infinity`
- There's a Math class: `Math.floor`, `Math.ceil`, etc.

Strings

```
let snack = 'coo';
snack += 'kies';
snack = snack.toUpperCase();
```

- Can be defined with single or double quotes
  - Many style guides prefer single-quote, but there is no functionality difference
- **Immutable**
- No char type: letters are strings of length one
- Can use plus for concatenation
- Can check size via `length` property (not function)

```
let abc = "abcdefghijklmnopqrstuvwxyz";
let esc = 'I don\'t \n know';   // \n new line
let len = abc.length;           // string length
abc.indexOf("lmno");            // position of 1st occurrence of string, -1 if doesn't contain
abc.lastIndexOf("lmno");        // position of last found occurrence
abc.slice(3, 6);                // extracts "def", negative values count from behind
abc.replace("abc","123");       // find and replace, takes regular expressions
abc.toUpperCase();              // convert to upper case letters
abc.toLowerCase();              // convert to lower case letters
abc.concat(" ", str2);          // abc + " " + str2
abc.charAt(2);                  // character at index 2: "c"
abc[2];                         // character at index 2: "c",  not fully implemented across all
                                // browsers, can't set the character using this notation, avoid
abc.charCodeAt(2);              // character code at index: "c" -> 99
abc.split(",");                 // splitting a string on commas gives an array
abc.split("");                  // splitting on every character
128.toString(16);               // number to hex(16), octal (8) or binary (2)
abc.trim();                     // removes whitespace from both ends of the string
abc.substring(4,6);             // extracts the characters, between two specified indices
                                // (not including ending char): "ef"
abc.substr(4,6);                // Extracts characters beginning at specified start position
                                // and through the specified number of characters: "efghij"
```

All string methods return a new value. They do not change the original variable.

Equality

- JavaScript's == and != are basically broken: they do an implicit type conversion before the comparison.

```
'' == '0'   // false
 '' == 0   // true
0 == '0'   // true
NaN == NaN   // false
 [''] == ''   // true
false == undefined   // false
  false == null   // false
null == undefined   // true
```

- Instead of fixing == and != , the ECMAScript standard kept existing behavior but added === and !==

```
'' === '0'   // false
 '' === 0   // false
0 === '0'   // false
NaN == NaN   // still weirdly false
 [''] === ''   // false
false === undefined   // false
  false === null   // false
null === undefined   // false
```

Always use === and !== and don't use == or !=

Functions

- One way of defining a JavaScript function is with the following syntax:

```JS
function name() {
   statement;
   statement;
   ...
   return ...
}
```

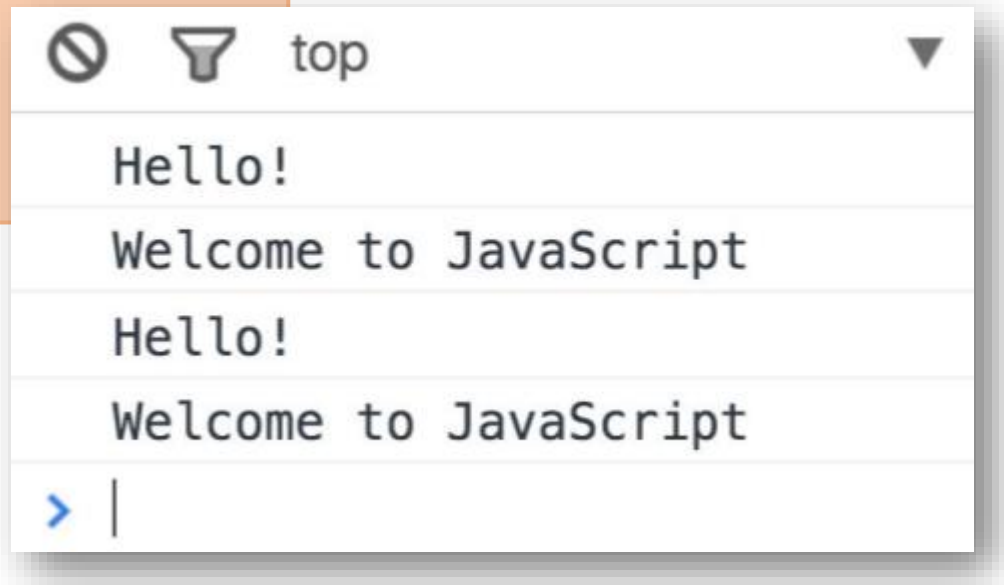
Function example

```
function hello() {
   console.log('Hello!');
   console.log('Welcome to JavaScript');
}

hello();
hello();
```

JS

Console output:



The browser "executes" the function definition first, but that just creates the hello function (and it doesn't run the hello function), similar to a variable declaration.

Function example

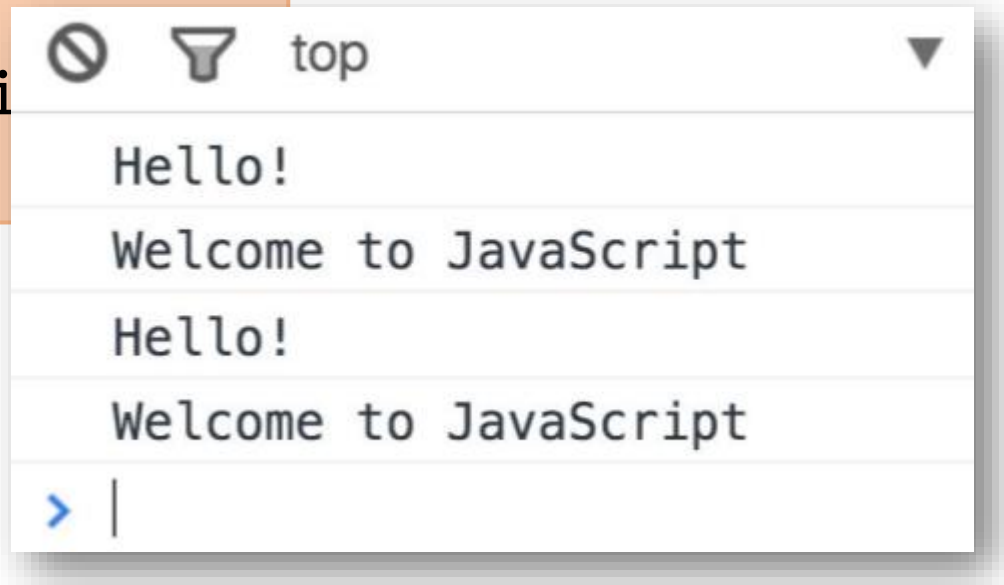
JavaScript Hoisting refers to **the process whereby the interpreter appears to move the declaration of functions, variables or classes to the top of their scope, prior to execution of the code**.

JS

```
hello();
hello();

function hello() {
   console.log('Hello!');
   console.log('Welcome to JavaScri
}
```

Console output:



- This works because function declarations are "hoisted" (mdn)
- Try not to rely on hoisting when coding. It gets bad.

Variable Hoisting

- Do you know what value will be printed in console if the following is executed as a JavaScript program?
- Why undefined? Recall that undefined is for a variables that **has been declared** but has not yet been assigned a value

```
console.log(foo); // undefined
var foo = 10;
```

JS

is actually interpreted like this:

```
var foo; // foo is hoisted
console.log(foo);
foo = 10;
```

JS

Variable Hoisting

- Do you know what value will be alerted if the following is executed as a JavaScript program?

JS

```
var foo = 1;
function bar() {
    alert(foo); // undefined
    if (!foo) {
        var foo = 10;
    }
    alert(foo); // 10
}
bar();
```

JS

```
var foo = 1;
function bar() {
    var foo;      // foo is hoisted
    alert(foo); // var has function scope
    if (!foo) {
        foo = 10;
    }
    alert(foo);
}
bar();
```

Variable Hoisting

- Do you know what value will be alerted if the following is executed as a JavaScript program?

JS

```
var foo = 1;
function bar() {
   alert(foo); // 1
   if (!foo) {
      let foo = 10;
   }
   alert(foo); // 1
}
bar();
```

- Let has block scope is not hoisted outside of its block
- foo has the value of 1 inside function (from the global foo assignment)
- Condition within the if statement is true, foo is assigned the value 10 within the block
- Outside the if statement block, foo maintains its global value

Arrays

- Arrays are Object types used to create lists of data.

```JS
// Creates an empty list
let list = [];
let groceries = ['milk', 'cocoa puffs'];
groceries[1] = 'kix';
```

  - 0-based indexing
  - Mutable
  - Can check size via length property (not function)

Arrays – Iterating through array

- You can use the familiar for-loop to iterate through a list:

```JS
let groceries = ['milk', 'cocoa puffs', 'tea'];
for (let i = 0; i < groceries.length; i++) {
   console.log(groceries[i]);
}
```

- Or use a for-each loop via `for...of` (mdn):
  (intuition: **for** each item **of** the groceries list)

```JS
let groceries = ['milk', 'cocoa puffs', 'tea'];
for (let item of groceries) {
   console.log(item);
}
```

```
let arr = [5,10,15];
let len = arr.length;      // array length: 3
arr[1];                    // second element of the array (index starts at 0): 10
arr.push(1);               // adds new item(s) to the end: [5,10,15,1]
arr.push(2,3);             // [5,10,15,1,2,3]
arr.pop();                 // removes last item and returns it's value, [5,10,15,1,2]
arr.shift();               // removes first element and returns it's value, [10,15,1,2]
arr.unshift(0);            // adds new item(s) to beginning, [0,10,15,1,2]
arr.reverse();             // reverses the array in place, [2,1,15,10,0]
arr.concat([9,8]);         // merges 2 or more arrays.Arrays unmodified. Returns new array: [2,1,15,10,0,9,8]
arr.slice(2,4);            // returns a copy of a portion of an array into a new array: [15,10]
arr.find(function check(i) { return i >= 5; });
                           // returns the value of the first element in array that pass test: 15
arr.findIndex(function check(i) { return i >= 5; });
                           // returns the index of the first element in array that pass test: 2
arr.filter(function check(i) { return i >= 5; });
                           // creates new array with every element in array that pass test: [15,10,9,8]
arr.indexOf(0)             // Search the array for an element and returns its position
arr.splice(start[, deleteCount[, item1[, item2[, ...]]]]);
                           // changes content by removing /or adding elements. start=Index at which to start
changing, deleteCount=number of array elements to remove, item1=element(s) to add
arr.splice(2,1,4,6);   // removes one element from index 2 and adds 4,6 in the same index [2,1,4,6,10,0,9,8]
delete arr[1];             // delete but not removes the element: [2,,4,6,10,0,9,8]. Use splice instead.
[5,10,15,1,2,3].sort();    // sort the elements of array in Unicode code point order: [1,10,15,2,3,5]

[5,10,15,1,2,3].sort(function compFunc(a,b) { return a-b; });

                           // For first iteration a=5 and b=10. If compFunc(a,b) > 0 : sort b to an index
                           // lower than a, else(<0 or ===0) the two elements will not change indexes
```

Objects

- Every JavaScript object is a collection of property-value pairs.
- Objects can be initialized using `new Object()`, `Object.create()`, or using the literal notation (initializer notation).
- An object initializer is a comma-delimited list of zero or more pairs of property names and associated values of an object, enclosed in curly braces {} as shown below:

JS

```
// Creates an empty object
const prices = {};
// Non empty object
const scores = { 'peach': 100, 'mario': 88, 'luigi': 91 };
```

Objects literal notation

```JS
// Creates an empty object
const prices = {};
// Non empty object
const scores = { 'peach': 100, 'mario': 88, 'luigi': 91 };
```

- There are two ways to access the value of a property:

1. `objectName[property]` `Ex: console.log(scores['peach']); // 100`
2. `objectName.property` (for string keys) `Ex: console.log(scores.peach);`

Objects – Adding property

- To add a property to an object, name the property and give it a value:

```JS
const scores = { peach: 100, mario: 88, luigi: 91 };
scores.toad = 72;
let name = 'super';
scores[name] = 102;
console.log(scores);
```

```
▶ {peach: 100, mario: 88, luigi: 91, toad: 72, super: 102}
```

Objects – Deleting property

- To remove a property to an object, use delete:

```JS
const scores = { peach: 100, mario: 88, luigi: 91 };
scores.toad = 72;
let name = 'super';
scores[name] = 102;
delete scores.peach;
console.log(scores);
```

```
▶ {mario: 88, luigi: 91, toad: 72, super: 102}
```

Objects – Iterating through object

- Iterate through a map using a for...in loop (mdn):
  (intuition: **for** each key **in** the object)

```JS
for (key in object) {
   // … do something with object[key]
}

for (let name in scores) {
   console.log(name + ' got ' + scores[name]);
}
```

You can't use `for...in` on lists; only on object types
You can't use `for...of` on objects; only on list types

Objects – How to transfer between client-server?

- Strings are lightweight and therefore very useful when transporting data.
  - Convert list (arrays) or objects to Strings
  - JavaScript Object Notation (JSON) is a standard text-based format for representing structured data based on JavaScript object syntax
- A JavaScript object can be easily converted to a JSON string using `JSON.stringify()` function
- JSON string can easily converted to JavaScript object using `JSON.parse()`

JS

```
const obj = {firstname : "Sam", lastname : "Shark", age : 41};
const jsonObj = JSON.stringify(obj);
console.log(jsonObj);
```

```
"{"firstname":"Sam","lastname":"Shark","age":41}"
```

Object to JSON – Examples

```
let contacts = {};
contacts["name"] = "Timothy";
contacts["age"] = 35;
contacts["address"] = {};
contacts["address"]["street"] = "1 Main St";
contacts["address"]["city"] = "Montreal";
contacts["interests"] = [];
contacts["interests"][0] = "cooking";
contacts["interests"][1] = "biking";
```

Create the whole JavaScript object once OR Create the JavaScript object as data become available

JS

```
let contacts = {
   name :"Timothy",
   age : 35,
   address : {
      street : "1 Main St",
      city : "Montreal"
   },
   interests:["cooking", "biking"]
};
```

```
let contacts = {};
contacts.name = "Timothy";
contacts.age = 35;
contacts.address = {};
contacts.address.street = "1 Main St";
contacts.address.city = "Montreal";
contacts.interests = [];
contacts.interests[0] = "cooking";
contacts.interests[1] = "biking";
```

The same

```
jsonStr = JSON.stringify(contacts);
```

```
"{"name":"Timothy","age":35,"address":{"street":"1 Main St","city":"Montreal"},"interests":["cooking","biking"]}"
```

```
conObj = JSON.parse(jsonStr);
```

JSON.parse()

- When using the `JSON.parse()` on a JSON derived from an array, the method will return a JavaScript array, instead of a JavaScript object.

JS

```
const text = '[ "Ford", "BMW", "Audi", "Fiat" ]';
const obj = JSON.parse(text);
console.log(obj);
```

```
["Ford", "BMW", "Audi", "Fiat"]
```

- Date objects are not allowed in JSON. If you need to include a date, write it as a string. You can convert it back into a date object later:

JS

```
var text = '{ "name":"John", "birth":"1986-12-14", "city":"New York"}';
var obj = JSON.parse(text);
obj.birth = new Date(obj.birth);
```

JSON.parse()

- You can use the second parameter, of the `JSON.parse()` function, called reviver.
- The reviver parameter is a function that **checks each property**, before returning the value; we can use a function to process each value

JS

```
var text = '{ "name":"John", "birth":"1986-12-14", "city":"New York"}';
var obj = JSON.parse(text, function (key, value) {
  if (key == "birth") {
    return new Date(value);
  } else {
    return value;
  }
});
```