JavaScript I

(JS)

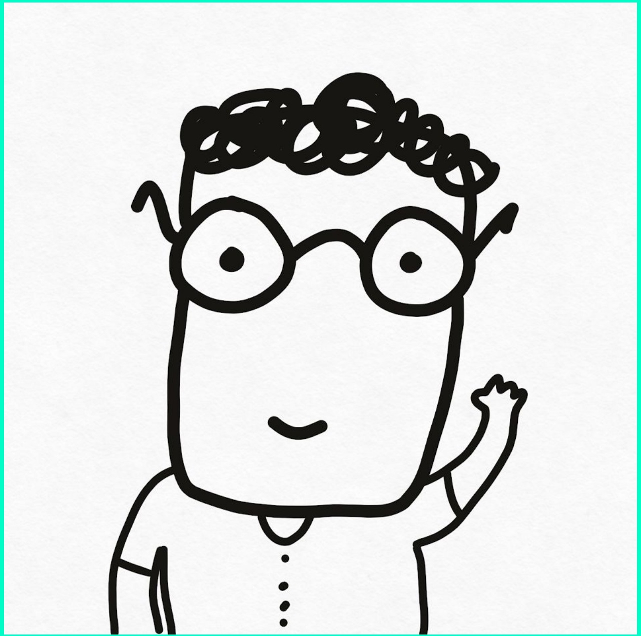
Lesson 1

Original by Alexandra Yamaui, edited by Rich Steinmetz

Rich Steinmetz

- developer for 5 years 💻
- career changer🏃
- blogger (richstone.io)🔮
- father 🍼

- Motivation: Create 🔮🔮



You

- Your name :)
- What are you doing currently?
- Why do you want to get into coding?
- How do you feel about coding?

- What's your Operating System (Windows, Mac, Linux)
- What do you know about JS and the terminal?

Why JavaScript?

- JS is eating the world
- JS is everywhere
- JS can do all the things

What's JavaScript?

- HTML
- CSS (styles)

```
<!DOCTYPE html>
<html>
        <head>
                <title>Page Title</title>
        </head>
        <body>

                <h1>This is a Heading</h1>
                <p>This is a paragraph.</p>
                <div class='item', id='div-tag'>
                        This is an item
                </div>
                <a href='https://migracode.eu/'>
                        Link to Migracode website
                </a>

        </body>
</html>
```

What's JavaScript?

Javascript

- Language for webpage manipulation
- Add new HTML to the page
- Change the existing content
- Modify styles (CSS)
- React to user actions, like on mouse clicks, pointer movements, key presses.

```
<!DOCTYPE html>
<html>
      <head>
            <title>Page Title</title>
      </head>
      <body>

            <h1>This is a Heading</h1>
            <p>This is a paragraph.</p>
            <div class='item', id='div-tag'>
                  This is an item
            </div>
            <a href='https://migracode.eu/'>
                  Link to Migracode website
            </a>

      </body>
</html>
```

What's JavaScript?

Javascript

- Send requests over the network to remote servers, download and upload files (so-called AJAX and COMET technologies).
- Get and set cookies, ask questions to the visitor, show messages.
- Remember the data on the client-side ("local storage").

```
<!DOCTYPE html>
<html>
      <head>
            <title>Page Title</title>
      </head>
      <body>

            <h1>This is a Heading</h1>
            <p>This is a paragraph.</p>
            <div class='item', id='div-tag'>
                  This is an item
            </div>
            <a href='https://migracode.eu/'>
                  Link to Migracode website
            </a>

      </body>
</html>
```

How does JavaScript work?

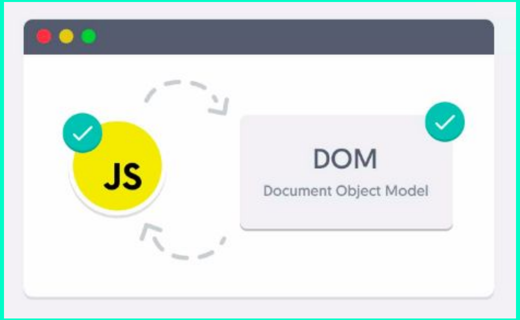
The Browser has an embedded **JavaScript engine**

- Program that executes JavaScript code
- Runs along with the **rendering engine** via the Document Object Model (**DOM**).

How does JavaScript work?

DOM (Document Object Model)

- Structural representation of the HTML document in the browser
- The way JavaScript interacts with the HTML and CSS



How does javascript work?

We embed JavaScript code inside HTML documents enclosing the code inside **<script>** tags

```
<!DOCTYPE html>
<html>
      <head>
            <title>Page Title</title>
            <script>
                  Javascript code
            </script>
      </head>
      <body>

            <h1>This is a Heading</h1>
            <p>This is a paragraph.</p>
            <div class="item" id="div-tag">
                  This is an item
            </div>
            <a href="https://migracode.eu/">
                  Link to Migracode website
            </a>

      </body>
</html>
```

How does JavaScript work?

We embed JavaScript code inside HTML documents enclosing the code inside **<script>** tags

```
<!DOCTYPE html>
<html>
      <head>
            <title>Page Title</title>
            <script>
                  console.log("Hello World!");

                  console.log("Hello");

                  console.log("in Migracode");
            </script>
      </head>
      <body>
            …
      </body>
</html>
```

How does JavaScript work?

We embed JavaScript code inside HTML documents enclosing the code inside **<script>** tags

```
<!DOCTYPE html>
<html>
      <head>
            <title>Page Title</title>
            <script src="example.js"></script>
      </head>
      <body>
            …
      </body>
</html>
```

How does JavaScript work?

Script:

- Sequence of instructions
- Every instruction is executed in order (from top to bottom) by the execution thread
- Every instruction is terminated in semicolon (;)

example.js

```
console.log("Hello World!");

console.log("I’m Learning JavaScript...");

console.log("in Migracode");
```

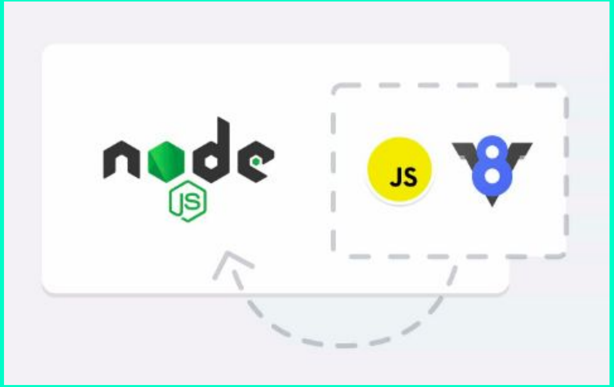
Exercise 0

- open your browser (like Chrome or Firefox)
- right-click >> Inspect
- type `console.log("Hello World");`

What's Node JS?

Node JS

- JavaScript engine outside the browser
- You can run JS in a browser
- You can run JS(node) on any computer & server **without a browser**

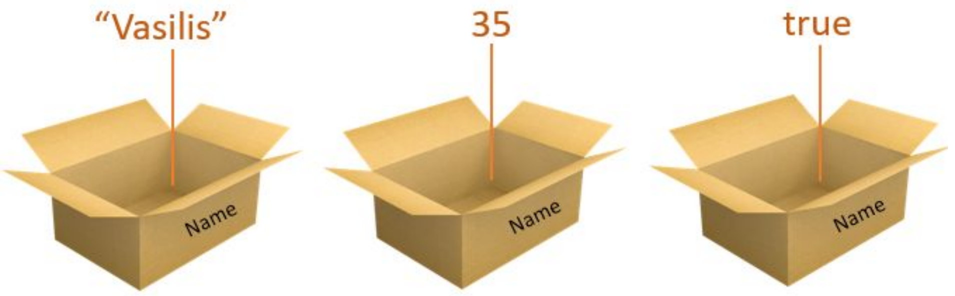


JavaScript Vocabulary & Syntax

{({ [{}]: { [{}]: {} } }[{}][{}]);}

JS Variables

Containers for storing data values



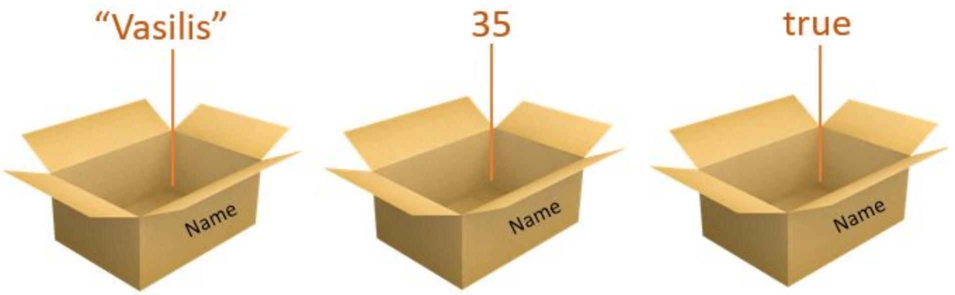
example.js

```
let variable1 = "Vasilis";

console.log(variable1);

variable1 = 35;
variable1 = true;
```

JS Variables

There are three keywords to define a variable:

- const
- let
- var

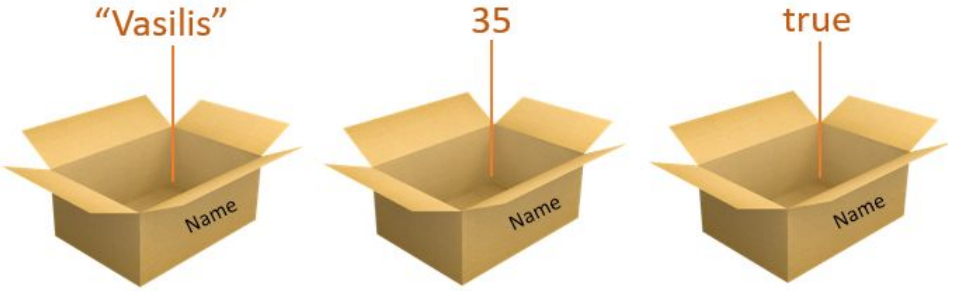


example.js

```
const constantVariable = "constant value";

console.log(constantVariable);


let rewritableVariable = "dynamic value";

console.log(rewritableVariable);


var rewritableVariable = "dynamic value";

console.log(rewritableVariable);
```

JS Variables

- **const**
    - Its value cannot change
    - If try to replace its value an Error will be thrown

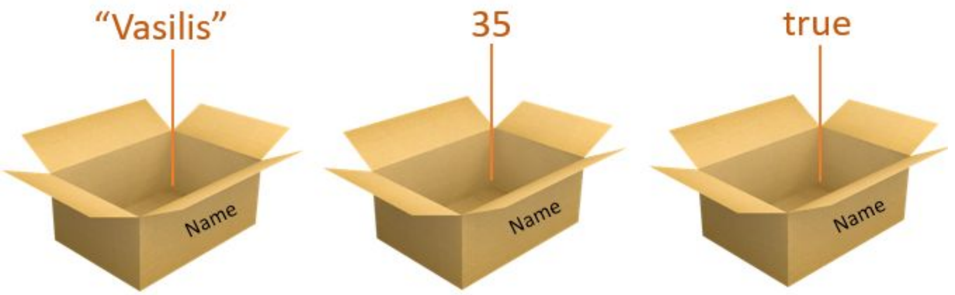


example.js

```
const constantVariable = "constant value";

console.log(constantVariable);


let rewritableVariable = "dynamic value";

console.log(rewritableVariable);


var rewritableVariable = "dynamic value";

console.log(rewritableVariable);
```

JS Variables

- **let**
  - Its value can be replaced with a different value

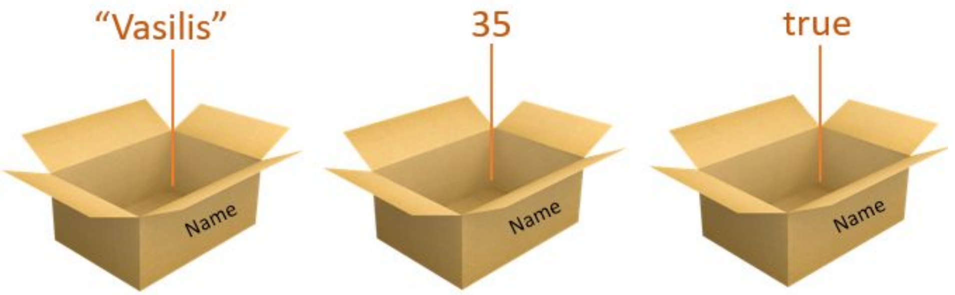


example.js

```
const constantVariable = "constant value";

console.log(constantVariable);


let rewritableVariable = "dynamic value";

console.log(rewritableVariable);


var rewritableVariable = "dynamic value";

console.log(rewritableVariable);
```

JS Variables

- **var**
  - Its value can be replaced with a different value

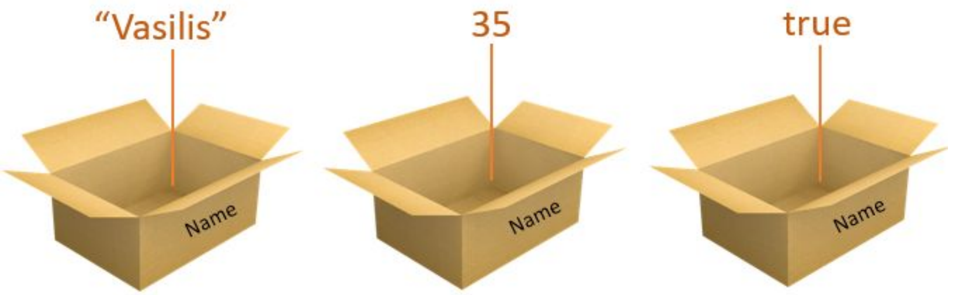


example.js

```
const constantVariable = "constant value";

console.log(constantVariable);


let rewritableVariable = "dynamic value";

console.log(rewritableVariable);


var rewritableVariable = "dynamic value";

console.log(rewritableVariable);
```

JS Variables

- **let vs var**
  - They have different scopes (we'll talk about this later)
  - Is preferable to use **let** to avoid unexpected errors

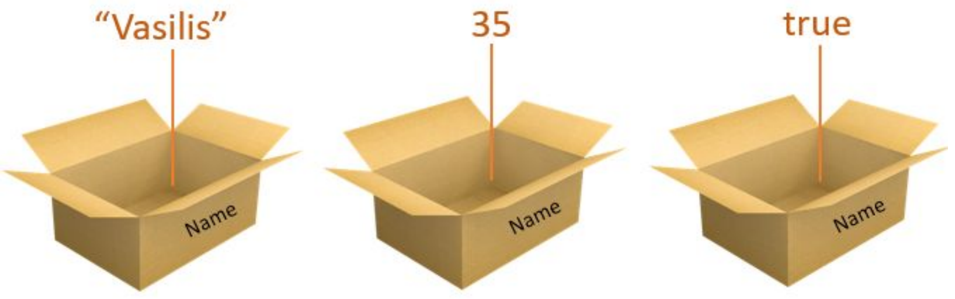


Why using const or let?

- previous to ES5 there vas only var
- many people still just use var
- const and let make your intent more clear

JS Data Types

- Number
- String
- Float
- Boolean
- (Function)
- Array
- Object
- Undefined
- Null



example.js

```
let name = "Vasilis"; // String type
console.log(name);            ^- Comment
let name = 35; // Number type
console.log(name);
let name = true; // Boolean type
console.log(name);
```

JS Data Types

- To know the type of a variable we can use the typeof operator

example.js

```
const message = "This is a string";

const messageType = typeof message;

console.log(messageType); // logs ?
```

JS Strings

- Sequence of characters
- Inside quote marks (can be simple quotes (‘) or double quotes (“))

example.js

```
const message = "This is a What?";
```

JS Strings

String concatenation:

- Two strings can be added using the concatenation operation (+)

example.js

```js
const greetingStart = "Hello, my name is ";
const name = "Alexandra";

const greeting = greetingStart + name;

console.log(greeting);
// Logs "Hello, my name Alexandra"
```

JS Numbers

- Don't need to be inside quotes
- Represent integer numbers

Operators:

- Sum (+)
- Difference (-)
- Division (/)
- Product (*)

example.js

```
const age = 30;


const sum = 10 + 2; // 12
const product = 10 * 2; // 20
const division = 10 / 2; // 5
const difference = 10 - 2; // 8
```

JS Floats

- Represent numbers with decimals (floating point numbers)

Operators:

- Sum (+)
- Difference (-)
- Division (/)
- Product (*)

example.js

```
const preciseAge = 30.612437;
```

JS Floats

- Round a float

example.js

```js
const preciseAge = 30.612437;
const roughAge = Math.round(preciseAge);
// ?
```

JS Libraries

- Pre-written JavaScript code
- It could be preloaded
- Can be downloaded from internet
- Ex: **Math** library is already preloaded, so we can use it right away

example.js

```
const preciseAge = 30.612437;
const roughAge = Math.floor(preciseAge);
// ?
```

JS DECLARING VS INITIALIZING VARIABLES

In order to assign a value to a variable we first need to **declare** it and then **initialize** it with a value. But this can also be done in a single line.

example.js

```
// Declaring a variable
let x;

// Initializing a variable
x = 2;

// Declaring and initializing a variable
let x = 2;
```

Exercises B - G

https://github.com/Migracode-Barcelona/javascript-module-1

JS Functions

example.js

```
const greetingStart = "Hello, my name is ";
const name = "Alexandra";
const greeting = greetingStart + name;
console.log(greeting);
```

JS Functions

example.js

```
const greetingStart = "Hello, my name is ";
const name = "Alexandra";
const greeting = greetingStart + name;
console.log(greeting);

const name = "Daniel";
const greeting = greetingStart + name;
console.log(greeting);

const name = "Ana";
const greeting = greetingStart + name;
console.log(greeting);
```

JS Functions

Duplication 😱

example.js

```
const greetingStart = "Hello, my name is ";
const name = "Alexandra";
const greeting = greetingStart + name;
console.log(greeting);

const name = "Daniel";
const greeting = greetingStart + name;
console.log(greeting);

const name = "Ana";
const greeting = greetingStart + name;
console.log(greeting);
```

JS Functions

Function:

- Block of reusable code designed to perform a particular task
- First we need to **define** a function and later **invoke** it (call it)
- Ex: console.log() is a built-in function

example.js

```
const greetingStart = "Hello, my name is ";
const name = "Alexandra";
const greeting = greetingStart + name;
console.log(greeting);

const name = "Daniel";
const greeting = greetingStart + name;
console.log(greeting);

const name = "Ana";
const greeting = greetingStart + name;
console.log(greeting);
```

JS Functions

Function definition:

- A function is defined typing the keyword function, followed by a name for the function, followed by parenthesis
- Inside of parenthesis go the parameters of the function

example.js

```
function functionName(parameter1, param2, ...) {
  // code to be executed
}
```

JS Functions

Block:

A block is group of instructions that are be executed together

example.js

```
function functionName(parameter1, param2) {

  // code to be executed

}
```

Block

JS Functions

Function definition:

- A function is defined typing the keyword function, followed by a name for the function, followed by parenthesis
- Inside of parenthesis go the parameters of the function

example.js

```
function functionName(parameter1, param2, ...) {
  // code to be executed
}


const greetingStart = "Hello, my name is ";
const name = "Alexandra";
const greeting = greetingStart + name;
console.log(greeting);
```

JS Functions

Function definition:

- A function is defined typing the keyword function, followed by a name for the function, followed by parenthesis
- Inside of parenthesis go the parameters of the function

example.js

```
function functionName(parameter1, param2, ...) {
  // code to be executed
}

const greetingStart = "Hello, my name is ";
const name = "Alexandra";
const greeting = greetingStart + name;
console.log(greeting);

function greetingFcn(...) {




}
```

JS Functions

Function definition:

- A function is defined typing the keyword function, followed by a name for the function, followed by parenthesis
- Inside of parenthesis go the parameters of the function

example.js

```
function functionName(parameter1, param2, ...) {
  // code to be executed
}


const greetingStart = "Hello, my name is ";
const name = "Alexandra";
const greeting = greetingStart + name;
console.log(greeting);


function greetingFcn(...) {
        const greetingStart = "Hello, my name is ";
        const name = "Alexandra";
        const greeting = greetingStart + name;
        console.log(greeting);
}
```

JS Functions

Function definition:

- A function is defined typing the keyword function, followed by a name for the function, followed by parenthesis
- Inside of parenthesis go the parameters of the function

example.js

```
function functionName(parameter1, param2, ...) {
  // code to be executed
}



function greetingFcn(...) {
      const greetingStart = "Hello, my name is ";
      const name = "Alexandra";
      const greeting = greetingStart + name;
      console.log(greeting);
}
```

JS Functions

Function definition:

- A function is defined typing the keyword function, followed by a name for the function, followed by parenthesis
- Inside of parenthesis go the parameters of the function

example.js

```
function functionName(parameter1, param2, ...) {
  // code to be executed
}


function greetingFcn(...) {
      const greetingStart = "Hello, my name is ";
      const name = "Alexandra";
      const greeting = greeting Start + name;
      console.log(greeting);
}
function greetingFcn(name) {
      const greetingStart = "Hello, my name is ";
      const greeting = greetingStart + name;
      console.log(greeting);
}
```

JS Functions

Function definition:

- A function is defined typing the keyword function, followed by a name for the function, followed by parenthesis
- Inside of parenthesis go the parameters of the function

example.js

```
function functionName(parameter1, param2, ...) {
  // code to be executed
}



function greetingFcn(name) {
      const greetingStart = "Hello, my name is ";
      const greeting = greetingStart + name;
      console.log(greeting);
}
```

JS Functions

Function definition:

- A function is defined typing the keyword function, followed by a name for the function, followed by parenthesis
- Inside of parenthesis go the parameters of the function

example.js

```
function functionName(parameter1, param2, ...) {
  // code to be executed
}



function greetingFcn(name) {
      const greetingStart = "Hello, my name is ";
      const greeting = greetingStart + name;
      console.log(greeting);
}
```

JS Functions

Function definition:

- A function is defined typing the keyword function, followed by a name for the function, followed by parenthesis
- Inside of parenthesis go the parameters of the function

example.js

```
function functionName(parameter1, param2, ...) {
  // code to be executed
}



function greetingFcn(name, greetingStart) {
      const greeting = greetingStart + name;
      console.log(greeting);
}
```

are we done?

JS Functions

Function invocation:

- A function is invoked typing its name followed by the parenthesis () operator
- The order of the parameters matter

example.js

```
// Function definition
function greetingFcn(name, greetingStart) {
        const greeting = greetingStart + name;
        console.log(greeting);
}


// Function invocation
greetingFcn("Alexandra", "Hello, my name is ");
```

JS Functions

Function invocation:

- A function is invoked typing its name followed by the parenthesis () operator
- The order of the parameters matter

example.js

```
// Function definition
function greetingFcn(name, greetingStart) {
      const greeting = greetingStart + name;
      console.log(greeting);
}

// Function invocation
greetingFcn("Alexandra", "Hello, my name is ");
greetingFcn("Daniel", "Hello, I’m ");
greetingFcn("Ana", "Hi!, my name is ");
```

JS Functions

Return value:

- A function can return a value or not
- To return value we use the **keyword** return
- When the execution reaches the return keyword it automatically returns to the main execution thread
- We need to do something with the returned value

example.js

```
// Function definition
// This function doesn't return a value, it
// just prints to the console
function greetingFcn(name, greetingStart) {
        const greeting = greetingStart + name;
        console.log(greeting);
}



// Function invocation
greetingFcn("Alexandra", "Hello, my name is ");
```

JS Functions

The functions returned value can be assigned to a variable

example.js

```
// Function definition
// This function returns a value
function greetingFcn(name, greetingStart) {
	const greeting = greetingStart + name;
	return greeting;
}

// Function invocation
const resultGreeting = greetingFcn("Alexandra",
"Hello, my name is ");

console.log(resultGreeting);
```

JS Functions

The functions returned value can be assigned to a variable

example.js

```
// Function definition
// This function returns a value
function greetingFcn(name, greetingStart) {
        const greeting = greetingStart + name;
        return greeting;
}

// Function invocation
const resultGreeting = greetingFcn("Alexandra",
"Hello, my name is ");

console.log(resultGreeting);
```

JS Functions

The functions returned value can be assigned to a variable

example.js

```
// Function definition
// This function returns a value
function greeting(name, greetingStart) {
      const greeting = greetingStart + name;
      return greeting;
}

// Function invocation
const resultGreeting = "Hello, my name is Alexandra"


console.log(resultGreeting);
```

JS Functions

The functions returned value can be assigned to a variable

example.js

```
// Function definition
// This function returns a value
function greetingFcn(name, greetingStart) {
        const greeting = greetingStart + name;
        return greeting;
}

// Function invocation
const resultGreeting = greetingFcn("Alexandra",
"Hello, my name is ");

console.log(greetingFcn("Hello, my name is ",
"Alexandra"));
```

Exercises

Exercise H (20 minutes)

1. Create the file `exercise-H.js` script in the folder `week-1/InClass`
2. Design and create a function that:
   i. takes in more than one input
   ii. uses string concatenation
   iii. this means adding two strings together
   iv. performs some form of operation on a number
   v. uses `return` to return a **string**
3. Add a comment above your function to explain what it does
4. Call your function and run your script
5. What's the difference between a `return` and `console.log`?
6. When would you choose to use functions over the way we have been scripting so far?

JS Functions

The functions alter the execution thread of the script.

example.js

```
// Function definition
function greetingFcn(name, greetingStart) {
      const greeting = greetingStart + name;
      console.log(greeting);
}



console.log("First print");
// Function invocation
greetingFcn("Alexandra", "Hello, my name is ");
console.log("Last print");
```

JS Functions

The functions alter the execution thread of the script.

example.js

```
// Function definition
function greetingFcn(name, greetingStart) {
        const greeting = greetingStart + name;      3
        console.log(greeting);                      4
}



console.log("First print");                         1
// Function invocation
greetingFcn("Alexandra", "Hello, my name is ");     2
console.log("Last print");                          5
```

JS Nested Functions

We can call other functions inside a function (nested function)

example.js

```
// Define function
function getAgeInDays(age) {
        return age * 365;
}

// Define function
function createGreeting(name, age) {
        const ageInDays = getAgeInDays(age);
        const message = "My Name is " + name + " and
I was born over " + ageInDays + " days ago!";
        return message;
}
```

JS Nested Functions

We can call other functions inside a function (nested function)

example.js

```
// Define function
function getAgeInDays(age) {
        return age * 365;
}

// Define function
function createGreeting(name, age) {
        const ageInDays = getAgeInDays(age);
        const message = "My Name is " + name + " and 
I was born over " + ageInDays + " days ago!";
        return message;
}

// Invoke function
console.log(createGreeting("Alexandra", 31));
```

JS Nested Functions

We can call other functions inside a function (nested function)

example.js

```
// Define function
function getAgeInDays(age) {
        return age * 365;                                              3
}

// Define function
function createGreeting(name, age) {
        const ageInDays = getAgeInDays(age);                           2
        const message = "My Name is " + name + " and                   4
I was born over " + ageInDays + " days ago!";
        return message;                                                5
}

// Invoke function
console.log(createGreeting("Alexandra", 31));                          1
```

JS Nested Functions

We can call other functions inside a function (nested function)

example.js

```
// Define function
function getAgeInDays(age) {
        return age * 365;                                3
}


// Define function
function createGreeting(name, age) {
        const ageInDays = getAgeInDays(age);             2
        const message = "My Name is " + name + " and     4
I was born over " + ageInDays + " days ago!";
        return message;                                  5
}


// Invoke function
console.log(createGreeting("Alexandra", 31));          1/6
```

Exercises

Exercise I (20 mins)

1. Create the file `exercise-I.js` script in the folder `week-1/InClass`
2. Write a function that returns the year someone is born given their age as input
3. Using the answer from step 1, write a function that takes someone's name and age as input and returns a string that states the person's name and year they were born in a sentence

JS Scopes

Scope:

Defines where a variable is available (read/write)

- Global
- Function (local)
- Block (local)

example.js

```
let x = "global x";

function foo() {
    let y = "local y";
    console.log(x); // x is available
}

foo();

console.log(y); // ?

{
    let y = "local y";
    console.log(x); // x is available
}

console.log(y); // ?
```

JS Scopes

Scope:

Defines where a variable is available (read/write)

- Global
- Function (local)
- Block (local)

example.js

```
let x = "global x";

function foo() {
        let y = "local y";
        console.log(x); // x is available
}

foo();

console.log(y); // ERROR because undefined

{
        let y = "local y";
        console.log(x); // x is available
}

console.log(y); // ERROR because undefined
```

JS Scopes

Scope:

Defines where a variable is available

- Global
- Function (local)
- Block (local)

example.js

```
let x = "global x";

function foo() {
    let x = "local x";
    console.log(x); // ?
}

foo();

console.log(x); // ?

{
    let x = "local x";
    console.log(x); // ?
}

console.log(x); // ?
```

JS Scopes

Scope:

Defines where a variable is available

- Global
- Function (local)
- Block (local)

example.js

```
let x = "global x";

function foo() {
    const x = "local x";
    console.log(x); // local x
}

foo();

console.log(x); // global x

{
    const x = "local x";
    console.log(x); // local x
}

console.log(x); // global x
```

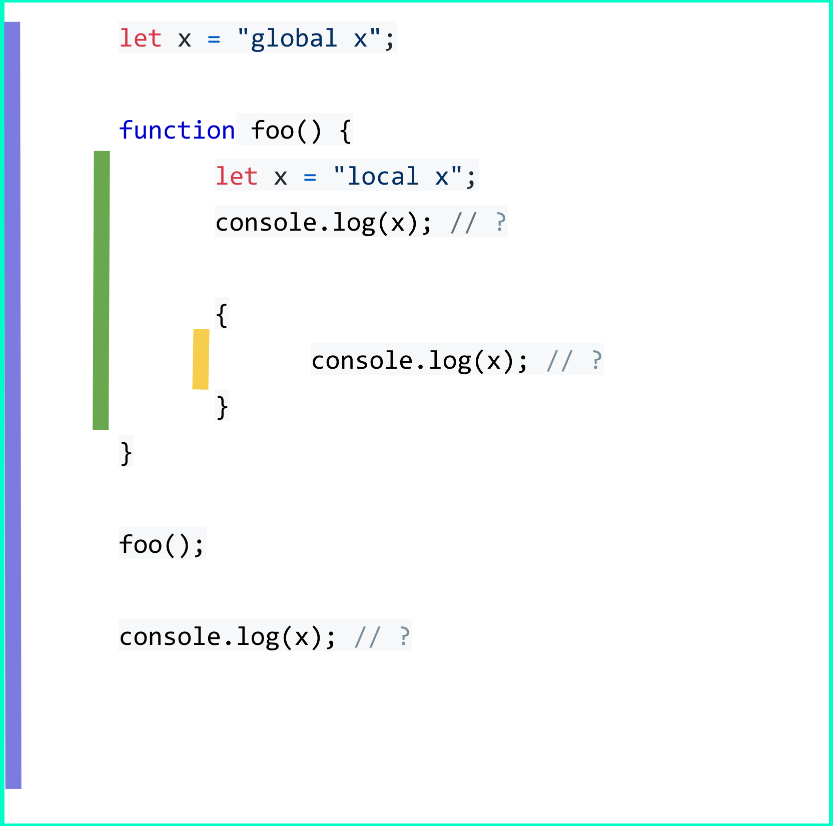
JS Scopes

Scope:

Defines where a variable is available

- Global
- Function (local)
- Block (local)

example.js



```
let x = "global x";

function foo() {
        let x = "local x";
        console.log(x); // ?

        {
                console.log(x); // ?
        }
}

foo();

console.log(x); // ?
```

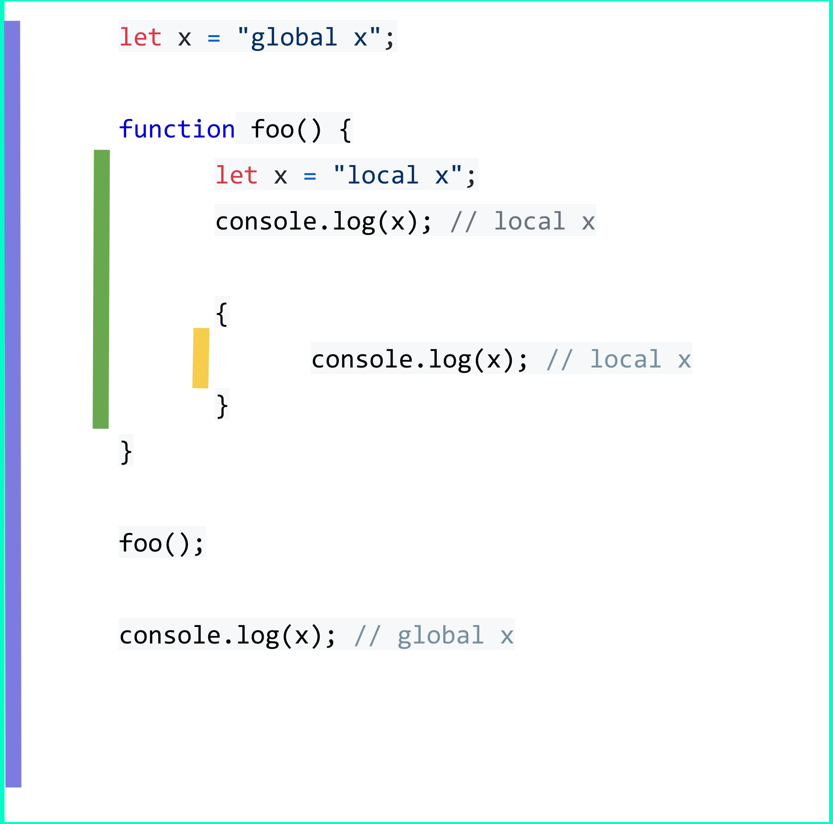
JS Scopes

Scope:

Defines where a variable is available

- Global
- Function (local)
- Block (local)

example.js



```
let x = "global x";

function foo() {
    let x = "local x";
    console.log(x); // local x

    {
        console.log(x); // local x
    }
}

foo();

console.log(x); // global x
```

JS Scopes

Scope:

Defines where a variable is available

- Global
- Function (local)
- Block (local)

example.js

```
let x = "global x";

function foo() {
    let x = "local x";
    console.log(x); // ?

    {
        let x = "nested local x";
        console.log(x); // ?
    }
    console.log(x); // ?
}

foo();

console.log(x); // ?
```

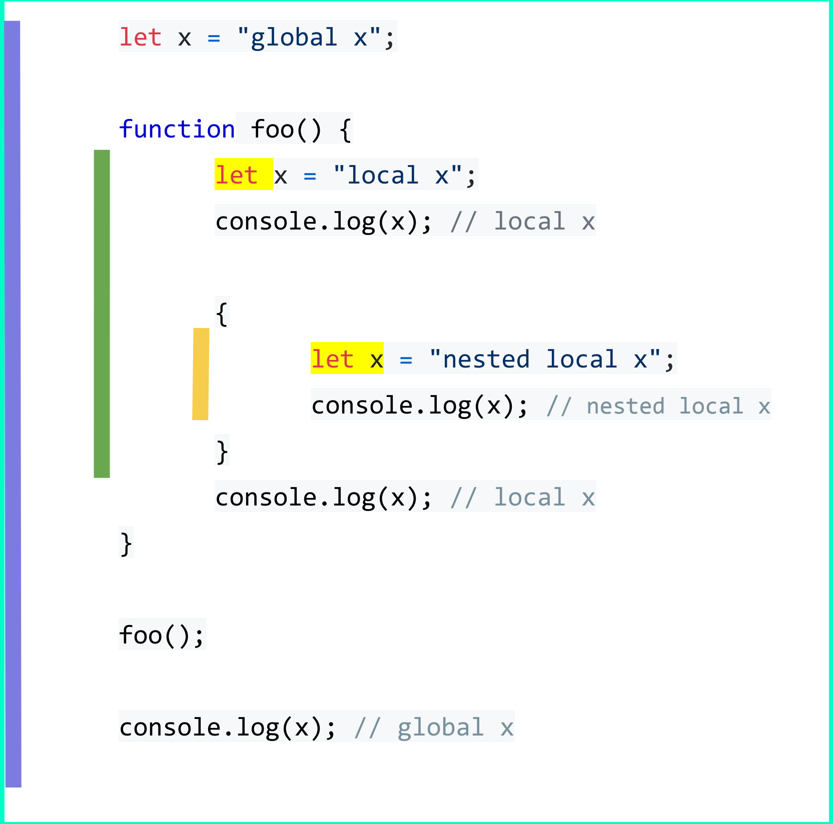
JS Scopes

Scope:

Defines where a variable is available

- Global
- Function (local)
- Block (local)

example.js



```
let x = "global x";

function foo() {
        let x = "local x";
        console.log(x); // local x

        {
                let x = "nested local x";
                console.log(x); // nested local x
        }
        console.log(x); // local x
}

foo();

console.log(x); // global x
```

JS Scopes

Scope:

Defines where a variable is available

- Global
- Function (local)
- Block (local)

example.js

```
let x = "global x"; //“local x”

function foo() {
    x = "local x";
    console.log(x); // ?

    {
        x = "nested local x";
        console.log(x); // ?
    }
}

foo();

console.log(x); // ?
```

JS Scopes

Scope:

Defines where a variable is available

- Global
- Function (local)
- Block (local)

example.js

```
let x = "global x";

function foo() {
    x = "local x";
    console.log(x); // local x

    {
        x = "nested local x";
        console.log(x); // nested local x
    }
}

foo();

console.log(x); // nested local x
```

JS Scopes

Scope (using var):

- Global
- Function (local)
- Block (local)

example.js

```
var x = "global x";

function foo() {
        var y = "local y";
        console.log(y); // local y
}

foo();

console.log(y); // ?

{
        var y = "local y";
        console.log(y); // local y
}

console.log(y); // ?
```

JS Scopes

Scope (using var):

- Global
- Function (local)
- Block (local)

example.js

```
var x = "global x";

function foo() {
        var y = "local y";
        console.log(y); // local y
}

foo();

console.log(y); // ERROR because undefined

{
        var y = "local y";
        console.log(y); // local y
}

console.log(y); // local y
```

JS Hoisting

JavaScript engine pushes (hoists) all the function definitions(and variables declared with var) to the top of the program

example.js

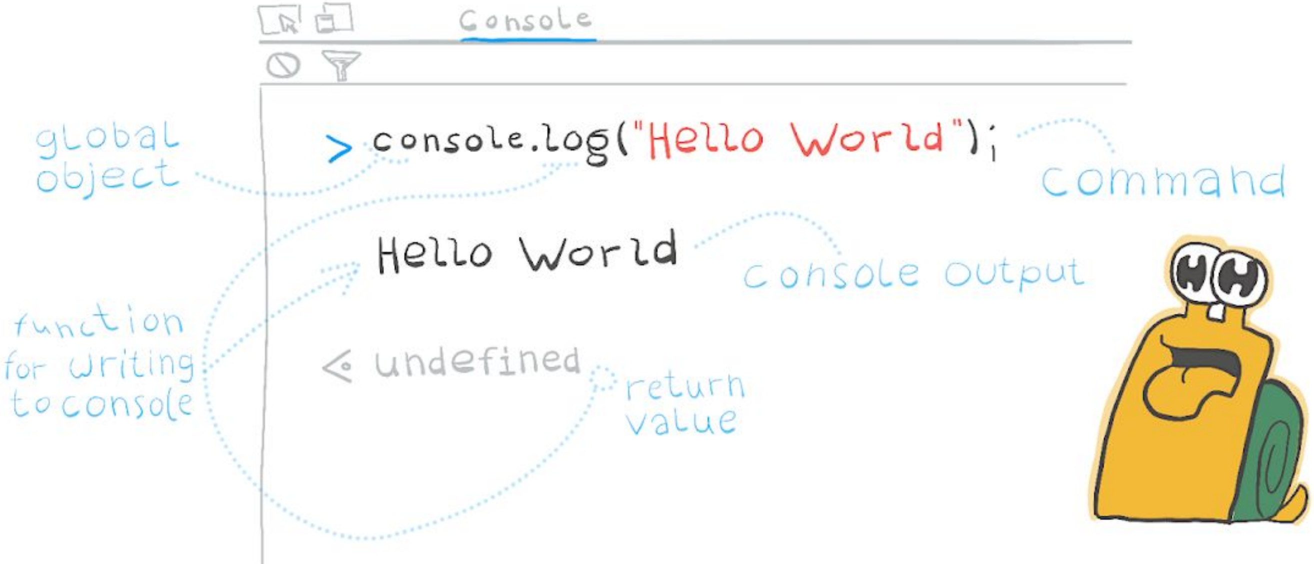
```
// Invoke function
greeting("Hello, my name is ", "Alexandra");


// Define function
function greeting(input_name, greeting) {
      var greeting = greetingStart + name;
      console.log(greeting);
}


x = "initializing before declaring x?"


var x;
```

First step to understand JS

Play with the language!

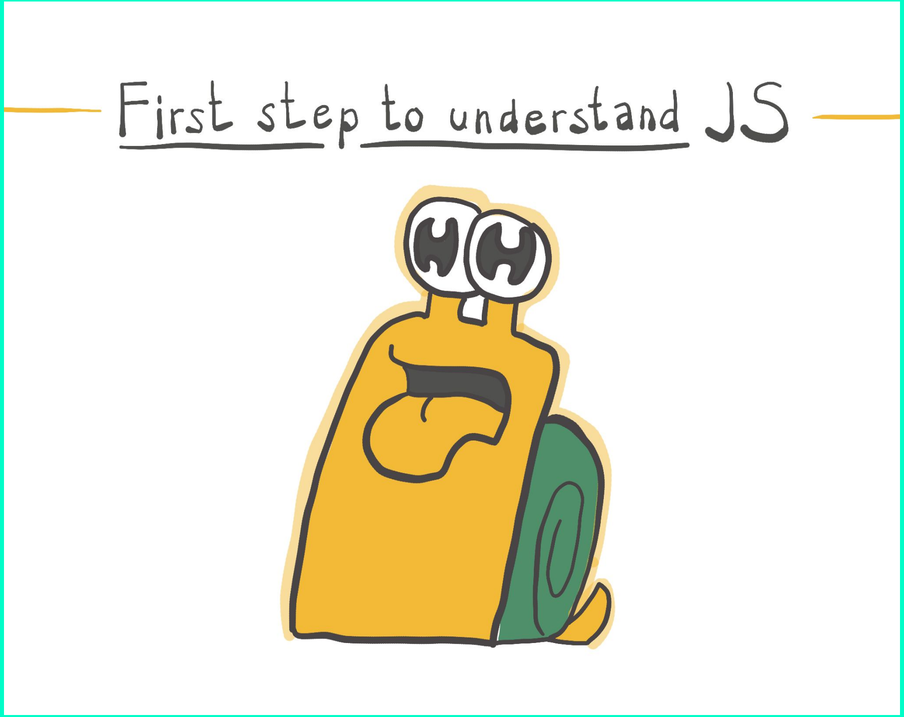
magic shortcut

or on mobile → jsconsole.com



```
> console.log("Hello World");
Hello World
< undefined
```

WHOA



```
function meetPerson(name, age) { // name and age are also variables
      const yearOfBirth = YOB(age);
      return “Meet ” + name + “, she was born in ” + yearOfBirth;
}


console.log(meetPerson(“Lola”, 38)); // “Lola” will be stored in name parameter (variable) and 38 will be stored in age
parameter
console.log(meetPerson(“Noura”, 14)); // “Noura” will be stored in name parameter (variable) and 14 will be stored in age
parameter
console.log(meetPerson(“Alexandra”, 31)); // “Alexandra” will be stored in name parameter (variable) and 31 will be stored
in age parameter
```