javascript essentials.js

About me

Bedis ElAcheche (@elacheche_bedis)

- Professional Scrum Master
- Lazy web & mobile developer
- FOSS lover
- Official Ubuntu Member (d4rk-5c0rp)
- Javascript fanatic
- I'm writing bugs into apps since 2008

What to expect ahead..

- What is Javascript?
- Language structure
- Conditions & Loops
- Functions
- Arrays & Objects
- Object oriented Javascript
- Asynchronous Javascript

What is Javascript?

- An interpreted programming langage

- Used to make webpages interactive

- Allows you to implement complex things on web pages

- The third layer of the cake of standard web technologies

What is Javascript?

HTML for content

CSS for presentation

Javascript for interactivity

What is Javascript?

Embedding javascript in a web page

```
<html>
    <head>
        <title>Hello Javascript</title>
    </head>
    <body>
        <script type="text/javascript">
            // your code should go here
        </script>
    </body>
</html>
```

What is Javascript?

Linking to a Javascript file

```
<html>
    <head>
        <title>Hello Javascript</title>
    </head>
    <body>
        <script type="text/javascript" src="app.js"></script>
    </body>
</html>
```

Variables: Assignment

Javascript is a dynamic language: when you declare a variable, you don't specify a type (and the type can change over time).

```
var firstName = "John";
var lastName = 'Doe';
var theAnswer = 42;
var aNumber = 22.5;
var aBoolean = true;
var aVariable;
var aNull = null;
var anArray = ['JS', null, 1337];
var anObject = {
    js: "JS Zero to Hero",
    html: "HTML5 & CSS3",
    version: 0
};
var aFunction = function(){};
```

Variables: Data Types

JavaScript defines 7 data types:

- Number
- Boolean
- String
- Object (Object, Date, Array)
- Function
- Undefined
- Null

Variables: Data Types

```
console.log(typeof firstName); // string
console.log(typeof lastName);  // string
console.log(typeof theAnswer); // number
console.log(typeof aNumber);   // number
console.log(typeof aBoolean);  // boolean
console.log(typeof aVariable); // undefined
console.log(typeof aNull);     // object
console.log(typeof anArray);   // object (arrays are objects in JS)
console.log(typeof anObject);  // object
console.log(typeof aFunction); // function
```

Variables: Type Conversion

Variables can be converted to a another data type:

- By the use of a Javascript function

- Automatically by Javascript itself

Variables: Type Conversion

```
// Converting Numbers to Strings
var number = 1.337;
console.log(String(number))             // '1.337'
console.log((number).toString())        // '1.337'
console.log(number.toExponential(2));   // '1.34e+0'
console.log(number.toFixed(4));         // '1.3370'
console.log(number.toPrecision(3));     // '1.34'

// Converting Booleans to Strings
console.log(String(false));    // 'false'
console.log(true.toString());  // 'true'

// Converting Null to Strings
console.log(String(null));         // 'null'
// Converting Undefined to Strings
console.log(String(undefined));    // 'undefined'
// Converting Functions to Strings
console.log(String(function(){ return 42; })); // 'function (){ return 42; }'
```

Variables: Type Conversion

```
// Converting Objects to Strings
console.log(String({}));        // '[object Object]'

// Converting Dates to Strings
console.log(String(new Date())); // 'Sun Oct 29 2017 19:42:33 GMT+0100 (CET)'
console.log(Date().toString());  // 'Sun Oct 29 2017 19:42:33 GMT+0100 (CET)'

// Converting Arrays to Strings
console.log(String([]));        // ''
console.log(String([42]));      // '42'
console.log(String([42,1337])); // '42,1337'
```

Variables: Type Conversion

```
// Converting Strings to Numbers
console.log(Number('13.37'));           // 13.37
console.log(Number(''));                // 0
console.log(Number('13 37'));           // NaN
console.log(+'42');                     // 42
console.log(+'13 37');                  // NaN
console.log(parseFloat('42'));          // 42
console.log(parseFloat('13.37'));       // 13.37
console.log(parseFloat('42 1337'));     // 42
console.log(parseFloat('42 answers')); // 42
console.log(parseFloat('answer 42'));   // NaN
console.log(parseInt('42'));            // 42
console.log(parseInt('13.37'));         // 13
console.log(parseInt('42 1337'));       // 42
console.log(parseInt('42 answers'));    // 42
console.log(parseInt('answer 42'));     // NaN
```

Variables: Type Conversion

```
// Converting Booleans to Numbers
console.log(Number(false));     // 0
console.log(Number(true));      // 1

// Converting Objects to Numbers
console.log(Number({}));            // NaN
// Converting Dates to Numbers
console.log(Number(new Date()));    // 1509303435369
// Converting Arrays to Numbers
console.log(Number([]));            // 0
console.log(Number([42]));          // 42
console.log(Number([42,1337]));     // NaN
// Converting Null to Numbers
console.log(Number(null));          // 0
// Converting Undefined to Numbers
console.log(Number(undefined));     // NaN
// Converting Functions to Numbers
console.log(Number(function(){}));  // NaN
```

Variables: Type Conversion

```
// Converting Strings to Booleans
console.log(Boolean('false')); // true
console.log(!!"false"); // true
// Converting Numbers to Booleans
console.log(!!42); // true
console.log(!!0); // false
// Converting Objects to Booleans
console.log(Boolean({}));           // true
// Converting Dates to Booleans
console.log(Boolean(new Date()));   // true
// Converting Arrays to Booleans
console.log(Boolean([]));           // true
// Converting Null to Numbers
console.log(Boolean(null));         // false
// Converting Undefined to Booleans
console.log(Boolean(undefined));    // false
// Converting Functions to Booleans
console.log(Boolean(function(){})); // true
```

Variables: Automatic Type Conversion

When JavaScript tries to operate on a "wrong" data type, it will try to convert the value to a "right" type.

```
console.log(42 + null);             // 42 because null is converted to 0
console.log(42 + '2');              // 422
console.log('42' + null);           // '42null' because null is converted to 'null'
console.log('42' + 2);              // 422 because 2 is converted to '2'
console.log('42' - 2);              // 40 because '42' is converted to 42
console.log('42' * '2');            // 84 because '42' and '2' are converted to 42 and 2
console.log(42 / '2');              // 21 because '2' is converted and 2
console.log('42' / '2');            // 21 because '42' and '2' are converted to 42 and 2
console.log('42' + '2');            // 422
console.log(42 + 2);                // 40
console.log((+'1337') + (+'42'));   // 1379
console.log((+'42s') + (+'42'));    // NaN
console.log(parseInt('42s') + 42); // 84
```

Variables: By value/By Reference

Scalar variables are passed by value

```
var foo = 42;
var bar = foo;
bar = 1337;
console.log(foo); // 42
console.log(bar); // 1337

var hello = 'Hello';
var world = hello;
console.log(world); // Hello
world = 'world';
console.log(hello)  // Hello
console.log(world)  // world
```

Variables: By value/By Reference

Dimensional variables are passed by reference

```
var johnDoe = { first : 'John', last : 'Doe', gender : 'male' };
var janeDoe = johnDoe;
janeDoe.first = 'Jane';
janeDoe.gender = 'female';
console.log(johnDoe); // { first: 'Jane', last: 'Doe', gender: 'female' }

var fruits = ['banana', 'orange', 'apple'];
var favorites = fruits;
console.log(favorites); // ['banana', 'orange', 'apple']
favorites[3] = 'tomato'; // yes tomato is a fruit :)
console.log(fruits); // ['banana', 'orange', 'apple', 'tomato']
```

Conditions: The 'if' statement

```
var name = 'Batman';
if (name === 'Batman') {
    name += ' rocks !';
} else if (name === 'Superman') {
    name += ' rocks too!!';
} else {
    name = 'Nevemrind!';
}
console.log(name); // 'Batman rocks !'
```

Conditions: Ternary operator '?'

```
var age = 42;
var accessAllowed;
if (age > 18) {
    accessAllowed = true;
} else {
    accessAllowed = false;
}
// the same
var accessAllowed = age > 18 ? true : false;
```

Conditions: Ternary operator '?'

```
var age = 42;
var message;
if(age < 3){
    message = 'Hi, baby!';
} else if(age < 18){
    message = 'Hello!';
} else if(age < 100){
    message = 'Greetings!';
} else {
    message = 'What an unusual age!';
}
// the same
var message =   age < 3 ? 'Hi, baby!' :
                age < 18 ? 'Hello!' :
                age < 100 ? 'Greetings!' : 'What an unusual age!';
```

Conditions: The 'switch' statement

```
switch (new Date().getDay()) {
    case 4:
    case 5:
        text = "Soon it is Weekend";
        break;
    case 0:
    case 6:
        text = "It is Weekend";
        break;
    default:
        text = "Looking forward to the Weekend";
}
```

Conditions: Operators

Greater than	>
Greater than or equal	>=
Less than	<
Less than or equal	<=
Equality	==
Inequality	!=
Identity / strict equality	===
Non-identity / strict inequality	!==
And	&&
Or	||

Conditions: Operators

```
var object1 = { value: 'key' };
var object2 = { value: 'key' };
var object3 = object2;
// Equality
console.log(42 == 42);             // true
console.log('42' == 42);           // true
console.log(42 == '42');           // true
console.log(0 == false);           // true
console.log(null == undefined);    // true
console.log(object1 == object2);   // false
console.log(object2 == object3);   // true
// Strict equality
console.log(42 === 42);            // true
console.log('42' === 42);          // false
console.log(0 === false);          // false
console.log(null === undefined);   // false
console.log(object1 === object2); // false
console.log(object2 === object3); // true
```

Conditions: Operators

```
var object1 = { value: 'key' };
var object2 = { value: 'key' };
var object3 = object2;
// Inequality
console.log(42 != 42);             // false
console.log('42' != 42);           // false
console.log(42 != '42');           // false
console.log(0 != false);           // false
console.log(null != undefined);    // false
console.log(object1 != object2);   // true
console.log(object2 != object3);   // false

// Strict inequality
console.log(42 !== 42);            // false
console.log('42' !== 42);          // true
console.log(0 !== false);          // true
console.log(null !== undefined);   // true
console.log(object1 !== object2); // true
```

Loops

```
// The For Loop
for (i = 0; i < 5; i++) {
}

// The For/In Loop
var person = {fname: 'John', lname:'Doe', age:25};
for (var property in person) {
    // loops through the properties of an object
}

// The While Loop
while (true) {
}

// The Do/While Loop
do {
} while(true);
```

Functions

```
// Function declaration
function addAB(a, b) {
    return a * b;
}

// Function expression
var sum = function(x, y) {
    return x + y;
};

console.log(addAB());       // NaN, You can't perform addition on undefined

console.log(sum(2, 3, 4)); // 5, added the first two; 4 was ignored
```

Functions : Self-Invoking Functions (IIFE)

- A self-invoking expression is invoked (started) automatically, without being called.

- Function expressions will execute automatically if the expression is followed by ().

- You cannot self-invoke a function declaration.

```
(function (person) {
    // I will invoke myself
    console.log('Hello ' + person);
})('World');
```

Functions : Scope

- Scope determines the accessibility (visibility) of variables.

- There are 2 scopes for variables:
    - The global (evil) scope
    - The local (function) scope

- A variable declared within a function is not accessible outside this function

- Unless using strict mode, it is not mandatory to declare variables (beware of typos...)

- Two scripts loaded from the same HTML page share the same global scope (beware of conflicts...)

Functions : Scope

```
var aVariableInGlobalScope;
// code here can use i
// code here can not use aVariableInFunctionScope

function myFunction() {
    // code here can use aVariableInGlobalScope
    var aVariableInFunctionScope;
    var anotherVariableInGlobalScope;
}

function myFunction2() {
    for (i = 0; i < 10; i++) {
        //i is in global scope!
    }
    for (var j = 0; j < 10; j++) {
        //j is in function scope!
    }
}
```

Functions : Scope

```
var a = 100;     // global scope
var b = a;       // global scope
console.log(a); // 100
b += 10;         // we add 10 to b using the unary operator
console.log(b); // 110
function addAB() {
    var b = 5; // local scope
    /*
        By preceding b with the var keyword it became a local
        variable to the function addAB() a in the outside scope
        is accessible by the function so is equal 100
     */
    return a + b;
}
console.log(addAB()); // 105
```

Functions : Scope

```
var a = 100; // global scope
var b = a;   // global scope
function addAB() {
    var b = 5; // b local scope
    function addIt() {
        var a = 95; // a local scope to addIt
        if (a > 90) {
            var b = 20;
        }
        /*
          Conditional blocks do not hold a scope
          So the new b is still accessible inside the addIt function
         */
        return a + b;
    }
    return addIt();
}
console.log(addAB()); // 115
```

Objects

- Objects are dynamic bags of properties

- It is possible to add and remove properties to an object at any time.

```
var person = { firstName: 'John', lastName: 'Doe' };
// Access a property
console.log(person.firstName); // John
// Dynamically add properties
person.gender = 'male';
person['age'] = 42;
// Remove a property
delete person.age;
// Check existence of a property
console.log(person.hasOwnProperty('gender')); // true
// Enumerate properties
for (var key in person) {
    console.log(key + ' : ' + person[key]);
}
```

Arrays

- Arrays are objects too

```
var cars = ['Mazda', 'Volvo', 'BMW'];
var cars = new Array('Mazda', 'Volvo', 'BMW');
console.log(cars.length); // 3
// Adds a new element 'Fiat' to cars
var newLength = cars.push('Fiat');
console.log(newLength); // 4
// Removes the last element 'Fiat' from cars
var fiat = cars.pop();
console.log(fiat); // 'Fiat'
// Removes the first element 'Mazda' from cars
var mazda = cars.shift();
console.log(mazda); // 'Mazda'
// Adds a new element 'Ferrari' at the beginning of cars
var newLength = cars.unshift('Ferrari');
console.log(newLength); // 3
```

Arrays

- Arrays are a special kind of objects, with numbered indexes

- Arrays use numbered indexes. Objects use named indexes

- Avoid new Array()

```
var points = new Array(40, 100, 1, 5, 25, 10); // Bad
var points = [40, 100, 1, 5, 25, 10]; // Good
var points = new Array(40, 100); // Creates an array with two elements (40 and 100)
var points = new Array(40); // Creates an array with 40 undefined elements ¯\_(ツ)_/¯
```

Object oriented Javascript

```
// Basic JavaScript objects with properties and a method
var johnDoe = {
    name: 'John Doe',
    speak: function() {
        return 'My name is John Doe';
    }
};
var johnSmith = {
    name: 'John Smith',
    speak: function() {
        return 'My name is John Smith';
    }
};
console.log(johnDoe.speak()); // 'My name is John Doe'
console.log(johnSmith.speak()); // 'My name is John Smith'
```

Object oriented Javascript

```
// Creating multiple objects of the same type with constructor functions.
function Person(name) {
    /*
    "this" allows you to reference a specific objects value
    without knowing the objects name
     */
    this.name = name;
    this.speak = function() {
        return 'My name is ' + this.name;
    }
}

// You call constructor functions with new
var johnDoe = new Person('John Doe');
var johnSmith = new Person('John Smith');
console.log(johnDoe.speak()); // 'My name is John Doe'
console.log(johnSmith.speak()); // 'My name is John Smith'
```

Object oriented Javascript : Prototype

- Every function has a prototype property that contains an object

- You can add properties and methods to the prototype object

- When you call for them to execute they are used just as if they belonged to the object

Object oriented Javascript : Prototype

```
function Animal(name, sound) {
    this.name = name;
    this.sound = sound;
}
// Use it to add a method
Animal.prototype.makeSound = function() {
    return this.name + " says " + this.sound;
};
var fox = new Animal('Duck', 'Quack');
console.log(fox.makeSound()); // Duck says Quack
var fox = new Animal('Fox', 'Ring-ding-ding-ding-dingeringeding');
// What does the fox say? :D
console.log(fox.makeSound()); // Fox says Ring-ding-ding-ding-dingeringeding
```

Object oriented Javascript : Private properties

- All properties in an object are public

- Any function can modify or delete these properties

- You can make properties private by declaring them as variables in a constructor

Object oriented Javascript : Private properties

```
function SecretCode() {
    // This value can't be accessed directly
    var secretNum = 42;
    // This function can access secretNum
    this.guess = function(num) {
        if (num === secretNum) { return 'You Guessed It'; }
        return 'Nop !';
    }
}
var secret = new SecretCode();
console.log(secret.secretNum);   // undefined
console.log(secret.guess(1337)); // 'Nop !'
// Even if we add another function it can't access the secretNum
SecretCode.prototype.getSecret = function() { return this.secretNum; };
console.log(secret.getSecret()); // undefined
```

Object oriented Javascript : Inheritance

- When we ask for a property if it isn't found in the main object then it is searched for in the prototype object.

- We are able to inherit methods and variables from any object in a chain of objects.

Object oriented Javascript : Inheritance

```
function Animal() {
    this.name = 'Animal';
    this.toString = function() {
        return 'My name is: ' + this.name;
    };
}
function Wolf() { this.name = 'Wolf'; }
// Overwrite the prototype for Wolf
Wolf.prototype = new Animal();
/*
    After you overwrite prototype its constructor points to the
    main object object so you have to reset the constructor after
 */
Wolf.prototype.constructor = Wolf;
var wolf = new Wolf();
// Wolf inherits toString from Animal
console.log(wolf.toString()); // 'My name is: Wolf'
```

Object oriented Javascript : Inheritance

```
// Overwrite drive parent method
Wolf.prototype.toString = function() {
    /*
    Call the parent method with apply so that the parent
    method can access the Trucks name value
     */
    return Animal.prototype.toString.apply(this) + ' and I\'m carnivore';
}
console.log(wolf.toString()); // 'My name is: Wolf and I'm carnivore'
```

Asynchronous Javascript

- One of the most important aspects of creating fluid HTML5 applications since Javascript is "single-threaded"

- Allows synchronization between all the different parts of the application :
  - Data extraction
  - Data processing and calculations
  - Rendering user interface elements
  - Animations
  - ...

Asynchronous Javascript : How to ?

- Events

- Callbacks

- Promises / Deferreds

Asynchronous Javascript : Events

- Can be something the browser does, or something a user does

- Event handler lets you execute code when events are detected

- Register an event handler for a given object

- Wait until the event is triggered

Asynchronous Javascript : Callbacks

- A callback function is essentially a pattern

- A function that is passed to another function as a parameter

- Callback is called at the completion of a given task; this prevents any blocking, and allows other code to be run in the meantime

Asynchronous Javascript : Callbacks Cons

- Error handling is easy to forget

- Sometimes it is common to have numerous levels of callback functions

- A messy code called «callback hell»

- The difficulty of following the code due to the many callbacks

- Hard to read, and hard to maintain code

Asynchronous Javascript : Deferreds/Promises

- A programming construct that have been around since 1976

- A way to organize asynchronous operations so that they appear synchronous

- A deferred represents work that is not yet finished

- A promise represents a value that is not yet known

Asynchronous Javascript : Promises

- A placeholder for a result which is initially unknown

- Promises can be returned and treated like a value, even though the value is not yet know.

Asynchronous Javascript : Deferreds

- A deferred is the work that must take place in order for a promise to “settle”

- May be in one of 3 possible states: fulfilled, rejected, or pending

- A deferred is settled if it’s not pending (it has been resolved or rejected)

- Once settled, a deferred can not be resettled.

```
slides.emit('Thank you!');
```