JavaScript for #NewDevs

Programming

Once you become a programmer, you will not only use the computer, but control it.

JavaScript is awesome (for beginners)

- no fancy setup needed
- is quite forgiving
- can be coded directly in the browser
- huge ecosystem (you don’t need to reinvent the wheel)

Browsers are awesome

- Run on your phone, computer, car and fridge
- Fancy APIs (application programming interface)
  - Bluetooth
  - User media (camera and mic)
  - Ambient Light
  - Vibration
  - Geolocation
  - USB
  - Device Motion

History and Facts

JavaScript History

- Developed in 1995 by Brendan Eich
- Development was done in 10 days
- Initial name was Mocha, first shipped as LiveScript
- First released in Netscape 2
- Was renamed to JavaScript in December 1995

Fun Fact #1

JavaScript is a trademark of Orcale.

Standardization

- JavaScript became an ECMA standard (ECMA-262) in 1997
- ECMAScript is often abbreviated as ES (with a version number)
- JavaScript is an implementation of ECMAScript
- ActionScript and JScript are other implementations of JavaScript

Version History

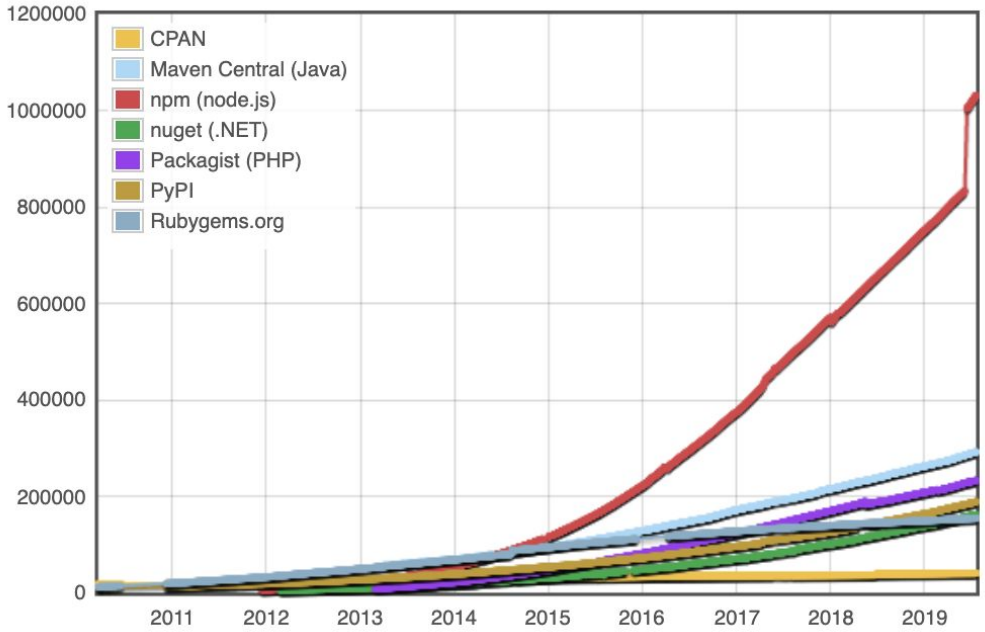
- ES1 released 1997
- ES2 released 1998
- ES3 released 1999
- ES4 was never released
- ES5 was released in 2009 (best browser support)
- ECMAScript 2015 was released in 2015
  - commonly known as ES6
  - Browser support is good, but it's still recommended to use ES5
- ES 2016, 2017, 2018 are defined, but only available via transpilers (Babel, TypeScript)

Modern JavaScript and Browser Support

- Older browsers may not understand modern JavaScript
- To support users we can use transpilers
- You can write modern JavaScript and the transpiler turns it into legacy JavaScript
- Popular transpilers are Babel and TypeScript

The JavaScript Ecosystem

- Over 1 million packages
- Huge community
- Open Source



Basic Concepts

console.log()

- console.log is your first debugging tool
- Put something in the parentheses to log it to the console

```
console.log('Hello World');
// ==> Hello World

console.log('I came this far');
// ==> I came this far

console.log('öakjdsöalksdjföljsödl');
// ==> 🙄🙄🙄
```

Basic arithmetic operators

- Doing basic math is no magic
- Operator precedence like in school
- Don’t divide by zero

```
console.log(1 + 2);
// ==> 3

console.log(1 + 2 * 5);
// ==> 11

console.log(0 / 0);
// ==> NaN 🤯🤯🤯
```

Variables #1

- Variables can store the result of an expression (e.g. the result of an arithmetic operation)
- Variables are declared with the keywords var, let and const
- Use `const` for variables that should never change

```
var result = 1 + 2;
console.log(result);
// ==> 3

const daysPerWeek = 7;
daysPerWeek = 5;
// ==> Uncaught TypeError: Assignment
to constant variable.
```

Variables #2

- Variable have arbitrary names, but
  - not whitespace
  - no dashes
  - not starting with a number
  - no reserved keywords
- UTF-8 characters
- Naming variables is one of the hardest challenges for professional developers 😪

```
var ॐ = "foo";

var jAdEnSmItHcAsInG = "oh noez!"

var ಠ_ಠ = "what?";
```

Basic data types

- JavaScript has seven basic data types
  - number
  - string
  - boolean
  - null
  - undefined
  - object
  - symbol

```
// Numbers
var positive = 1;
var negative = -1;

// String
var word = "foobar";

// Boolean
var truth = true;
var lie = false;

// Undefined
var nothing; // = undefined

// Null
var empty = null;
```

Objects

- Objects are create with two curly braces
- Objects are a key value store (like a dictionary)

```
// Empty object
var obj = {};

// Object with properties
var nico = {
    name: "Nico",
    website: "nico.codes"
}

// add properties
nico.age = 34;

// Read properties
console.log(nico.name); // ==> Nico
console.log(nico["name"]); // ==> Nico
```

Dynamically typed language

- The same variable can hold different data types during its lifecycle
- That's why JavaScript is a dynamically typed language

```
var foo;
// typeof(foo) ==> "undefined"

foo = 1;
// typeof(foo) ==> "number"

foo = "yolo!";
// typeof(foo) ==> "string"

foo = {};
// typeof(foo) ==> "object"

// JavaScript ¯\_(ツ)_/¯
```

Functions #1

- A function stores a bunch of code and can be executed any time
- Functions are defined with the `function` keyword and a name
- Use the name of the function with two round brackets to execute it

```
// Define a function
function sayHelloWorld() {
    console.log("Hello World");
}

// Execute the function
sayHello();

// ==> Hello World
```

Functions #2

- A function accepts arbitrary number of parameters
- Parameters are totally optional
- If you don't pass a parameter, it's value will be undefined

```
function greet(greeting, name) {
    var out = greeting + " " + name;
    console.log(out);
}

greet();
// ==> undefined undefined

greet("Hello");
// ==> Hello undefined

greet("Hello", "World");
// ==> Hello World
```

Functions #2

- A function can return a value
- The return keyword exits the function
- If no value is given, it will return undefined
- If a value is give, it will return the value or reference

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

var result = addNumbers(1,2);

console.log(result); // ==> 3
```

Call by value #1

- Variables that hold simple data types (string, number and boolean) use call by value
- If you assign a variable with a simple data type to another variable, only the value is passed and both hold the same value

```
var a = 1;
var b = a;
a = 2;

console.log(a); // ==> 2
console.log(b); // ==> 1
```

Call by value #2

- If you pass a variable with a simple data type into a function, it’s also call by value

```
var a = 4;

function square(a) {
    a = a * a;
}

console.log(a); // ==> 4
```

Call by reference

- Objects are handles with call by reference
- If a variable that holds an object is assigned to another variable, they both hold a reference to the same object
- The same applied if you pass a variable as an function argument

```
var foo = {};

var bar = foo;

bar.name = "Sara";

console.log(foo.name); // ==> Sara
```

if statements

- Used for conditional code execution
- Execute some code if something is true
- if statements try to evaluate the condition to a true or false

```
if(true) {
    console.log('yeah!');
}
// ==> yeah!

if(false) {
    console.log('no noez!');
}
// ==> condition is false, nothing
happens
```

if else statements

- Like if statement, but with an alternative code execution path
- If something is true do this, otherwise do that.

```
var name = "Sara";

if(name === "Sara") {
    console.log('Olá Sara!');
} else {
    console.log('Grüezi!');
}

// ==> Olá Sara!
```

Comparison operators

- Sometimes (e.g. if statements) values have to be compared
- JavaScript supports a strict and a type-converting comparison
- Equal (==) only compares value only
- Strict equal (===) compares value and type

```
if(1 == '1') {
    console.log('this is true');
}
// ==> this is true

if(1 === '1') {
    console.log('this is also true');
} else {
    console.log('no it\'s not!');
}
// ==> no it's not
```

Comparison operators

- Besides the equality operators, there are the inequality (!=) and strict-inequality operators (!==)

```
if(1 != '2') {
    console.log('this is true');
}
// ==> this is true

if(1 !== '1') {
    console.log('this is also true');
} else {
    console.log('no it\'s not!');
}
// ==> this is also true
```

Truthy and falsy

- Single values will evaluate to true or false in a boolean context (e.g. if statement)
- This is also known as truthy and falsy

Todo Example

Todo Example

Example

Further learning and resources

Resources

- JavaScript for kids
- Eloquent JavaScript
- ExploringJS