JS

github.com/INT-WAW

JAVASCRIPT

JAVASCRIPT

- JavaScript ist NICHT Java
- Bad Name, Shit happens
- Ehemals LiveScript, JScript
- Sprache des Web

ES5 / ES6

- Standardisierung durch ECMA Commitee
- Aktueller Standard: ES2015 (ES6)
- Schon heute ES2016 (ES7) nutzbar
- Funktionelle und Prototypische Sprache

GARBAGE COLLECTION

- Literale sind Runtime Memory
- Instanzen sind Heap Memory
- Garbage Collector loescht Heap Memory
- Vermeidung von Garbage ist geboten!

DATENTYPEN

PRIMITIVE

Primitive Datentypen sind `Boolean` , `Number` , `String` , `RegExp` . Primitive Datentypen verhalten sich anders als Objektinstanzen

PRIMITIVE: UNDEFINED

- `undefined` ist Default-Wert fuer Variablen
- `typeof undefined` ist `undefined`

BEISPIEL

```
var foo;
var bar = undefined;

console.log(foo === bar); // true
console.log(foo == bar);  // true

console.log(typeof foo);  // "undefined"
console.log(typeof bar);  // "undefined"
```

PRIMITIVE: NULL

- `null` ist ein leerer Wert
- `typeof null` ist `object` (ES5.1 Spezifikations-Bug)
- `typeof null` ist `null` (ES6 / ES7)

BEISPIEL

```
var foo = null;

console.log(foo === null); // true
console.log(typeof foo);   // "null" in ES6 and ES7
```

PRIMITIVE: BOOLEAN

- Literal ist `true` oder `false`
- Constructor ist `new Boolean(value)`
- `typeof 123` ist `boolean`
- Primitiver Datentyp

BEISPIEL

```
var foo = true;
var bar = new Boolean(true);

console.log(foo instanceof Object); // false
console.log(bar instanceof Object); // true

console.log(foo === true); // true
console.log(bar === true); // false

console.log(foo == true); // true
console.log(bar == true); // true

console.log(bar === bar);  // true, Object-Uniqueness
```

PRIMITIVE: NUMBER

- Literale sind `0123`, `0x123`, `123`, `1.23`, `0B0011`
- Constructor ist `new Number(value)`
- `typeof 123` ist `number`
- Primitiver Datentyp

BEISPIEL

```
var foo = 123;
var bar = new Number(123);
var qux = 0x7B;       // hex
var doo = 0B01111011; // binary

console.log(foo instanceof Object); // false
console.log(bar instanceof Object); // true

console.log(foo == bar);  // true
console.log(foo === bar); // false
console.log(foo === qux); // true
```

PRIMITIVE: STRING

- Literal ist `"foo"` oder `'foo'`
- Constructor ist `new String(value)`
- `typeof "foo"` ist `string`
- Primitiver Datentyp

BEISPIEL

```
var foo = "foo";
var bar = new String("foo");

console.log(foo instanceof Object); // false
console.log(bar instanceof Object); // true

console.log(foo == bar);  // true
console.log(foo === bar); // false
```

OBJEKTE UND ARRAYS

OBJEKTE

ALLE Objekte sind Unique und eindeutig identifizierbar, EGAL auf welche Art das Objekt instanziiert wurde.

OBJEKTE

- Literal ist `{}`
- Constructor ist `new Object()`
- Custom Constructor ist `Object.create(prototype,properties)`
- Objektbasierte Vererbung
- NICHT Objektorientierte Vererbung
- Prototype-Chain entscheidet Verhalten von `instanceof`

BEISPIEL

```
var foo = {};
var bar = new Object();

console.log(foo instanceof Object); // true
console.log(bar instanceof Object); // true

console.log(foo === {}); // false
console.log(bar === {}); // false

console.log(foo == {}); // false
console.log(bar == {}); // false
```

ARRAYS

ALLE Arrays leiten von Object ab und sind eindeutig identifizierbar, EGAL auf welche Art das Array instanziiert wurde.

ARRAYS

- Literal ist `[]`
- Constructor ist `new Array(length)`
- `typeof []` ist `object`

BEISPIEL

```
var foo = [];
var bar = new Array(1337);

console.log(foo[0]); // undefined
console.log(bar[0]); // undefined

console.log(foo.length); // 0
console.log(bar.length); // 1337
```

ARRAYS

- Array leitet von Object ab
- Jedes Array ist deshalb auch unique
- Prototype-Chain ist `Array` > `Object` > `null`

BEISPIEL

```
var foo = [];

var p1 = Object.getPrototypeOf(foo);
var p2 = Object.getPrototypeOf(p1);
var p3 = Object.getPrototypeOf(p2);

console.log(p1 === Array.prototype);  // true
console.log(p2 === Object.prototype); // true
console.log(p3 === null);             // true
```

FUNKTIONEN

FUNKTIONEN

- Definition via `function(parameter)`
- First-Class Functions
- `typeof (function(){})` ist `function`
- `new` keyword fuehrt zu `Object` Instanz
- Prototype-Chain ist `Function` > `Object` > `null`

BEISPIEL

```
var foo = function() { return 3 * 3; }; // GOOD
var bar = new Function('return 3 * 3'); // BAD

console.log(typeof foo); // 'function'
console.log(typeof bar); // 'function'

console.log(Object.prototype.toString.call(foo)); // '[object Function]'
console.log(Object.prototype.toString.call(bar)); // '[object Function]'

console.log(foo()); // 9
console.log(bar()); // 9
```

FUNKTIONSSCOPES

- Neuer Scope ist erstellt mit call Syntax `foo()`
- `new` keyword erstellt neue Objektinstanz
- `this` keyword referenziert auf aktuellen Scope
- `Function.prototype.call` fuer statische Scope Manipulation
- `Function.prototype.apply` fuer dynamische Scope Manipulation

BEISPIEL

```
var foo = function(a, b) {
        console.log(this.qux, a, b);
};
var bar = { qux: 'doo' }

foo.call(bar, 1, 2);                  // 'doo', 1, 2
foo.apply({ qux: 'woo' }, [ 1, 2 ]); // 'woo', 1, 2

foo(1, 2);     // undefined, 1, 2
new foo(1, 2); // undefined, 1, 2
```

CLOSURES

- Variablen (`var`) sind Function Scoped
- Jede Variable die nicht gebunden ist, ist frei zum Outer Scope

BEISPIEL

```
var foo = { bar: 1 };

var bar = function() {
        foo.bar++;
};

bar();
bar();

console.log(foo); // bar: 3
```

CLOSURES

- Bindung von Variablen durch Parameter
- Bindung von Variablen durch benamte Expressions

BEISPIEL

```
var foo = { bar: 1 };
var bar = function(foo) {
        foo.bar++;
};

var qux = { bar: 1 };

bar(qux);
bar(qux);

console.log(foo); // bar: 1
console.log(qux); // bar: 3
```

IIFE

- Immediately Invoked Function Expressions
- Sofort ausgefuehrte Funktionsausdruecke
- Typischerweise Bindung von Variablen
- Return Wert wird synchron ausgefuehrt

BEISPIEL

```
var foo = (function(a, b) {
        return a * b;
})(13, 37);

console.log(foo); // 481
```

JSON

JSON

- JavaScript Object Notation
- Interchange Datenformat fuer alle Programmiersprachen
- Alternative zu XML
- XML ist scheisse
- JSON ist awesome

BEISPIEL

```
var foo = {
        bar: true,
        qux: 123.23,
        doo: [ false, 'foo', 123 ]
};

var bar = JSON.stringify(foo);
var qux = JSON.parse(bar);
```