Javascript funktionen sind objekten

Von @binary_sequence (twitter)

Funktionen sind “first-class citizens”

1. Funktionen sind Objekten.
2. Funktionen haben Eigenschaften.
3. Funktionen können Variablen zugewiesen werden.
4. Funktionen können als Argument einer Funktion übergegeben werden.
5. Funktionen können aus einer Funktion zurückgegeben werden.
6. Kontext ist wichtig.

```
function f(){}
typeof(f); // "function"

let o = new Object();
typeof(o); // "object"

typeof(Object); // "function"

o = new f();
typeof(o); // "object"
```

1. Funktionen sind objekten

```
function f(){}

f.name; // "f"
f.constructor; // f Function()
{ [native code] }

typeof(Function); //
"function"

let g = new Function();
typeof(g); // "function"
g instanceof Object; // true
```

2. Funktionen haben Eigenschaften

```
function f() {
    return 1 + 1;
}

let g = f;

f(); // 2
g(); // 2
```

3. **Funktionen können Variablen zugewiesen werden**

```
function f() {
    return 1 + 1;
}

function g(arg) {
    return arg();
}

f(); // 2
g(f); // 2
g(); // Uncaught TypeError:
arg is not a function
```

4. **Funktionen können als Argument einer Funktion übergegeben werden**

```
function f() {
    function g() {
        return 1 + 1;
    }
    return g;
}

typeof(f); // "function"
typeof(g); // "undefined"

let h = f();
typeof(h); // "function"
h(); // 2

typeof(f()); // "function"
f()(); // 2
```

5. **Funktionen können aus einer Funktion zurückgegeben werden**

```
var a = 'main';

let o = new Object();
function f() {
   return this.a;
}
o.a = 'object o';
o.m = f;


o.m(); // 'object o'
f(); // 'main'
```

6. Kontext ist wichtig (Teil 1)

```
var a = 'main';

let o = {
    a: 'object o',
    m: function() {
        return this.a;
    }
};
let f = o.m;

o.m(); // 'object o'
f(); // 'main'
```

6. Kontext ist wichtig (Teil 2)

```
let o = {
    a: 'object o',
    m: function() {
        return this.a;
    }
};

function f(callback) {
    return callback();
}

o.m(); // 'object o'
f(o.m); // undefined
var a = 'main';
f(o.m); // 'main'
```

7. Kontext verlieren

```
var bgcolor = 'main';

let square = {
   bgcolor: '#11AAAA',
   change_color: function() {
       console.log(this.bgcolor);
   }
};

// <button id="btn">Change
color</button>
var btn =
document.getElementById('btn');

btn.addEventListener('click',
square.change_color);
```

8. Event

```
// Function
Function.bind(thisArg[, arg1[,
arg2[, ...]]])

let o = {
   a: 'object o',
   m: function() {
       return this.a;
   }
};

function f(callback) {
   return callback();
}

o.m(); // 'object o'
f(o.m.bind(o)); // 'object o'
```

9. Methode bind() (Teil 1)

```
function m() {
   return this.a;
}
let o = new Object();
o.a = 'object o';
o.m = m.bind(o);

function f(callback) {
   return callback();
}

o.m(); // 'object o'
f(o.m); // 'object o'
```

9. Methode bind() (Teil 2)

```
// function.call( thisArg[, arg1[, arg2[,
...]]] )

function Pet(name, age) {
    this.name = name;
    this.age = age;
}

function Dog(name, age) {
    this.race = 'Huskie';
}

function Cat(name, age) {
    Pet.call(this, name, age);
    this.race = 'Persian';
}

let p = new Pet('Boby', '5');
console.log(p.name); // Boby

let d1 = new Dog('Charlie', '7');
console.log(d1.name); // undefined

let d2 = new Cat('Mino', '4');
console.log(d2.name); // Mino
```

10. Methode call().

```
// Function.apply(thisArg, [argsArray])

let pets = ['Boby', 'Charlie', 'Mino'];
let wild = ['Tiger', 'Shark'];
let animals = pets.slice();

animals.push(wild); // 4
console.log(animals); // ['Boby',
'Charlie', 'Mino', ['Tiger', 'Shark']]

animals = pets.slice();
animals.push(wild[0]); // 4
animals.push(wild[1]); // 5
console.log(animals); // ['Boby',
'Charlie', 'Mino', 'Tiger', 'Shark']

animals = pets.slice();
animals.push.apply(animals, wild); // 5
console.log(animals); // ['Boby',
'Charlie', 'Mino', 'Tiger', 'Shark']
```

11. Methode apply().

Javascript Funktionen sind Objekten

By @binary_sequence (twitter)

“Javascript ist Perl im Front-End”