20.3 OOP Konstruktor

Man kann sich einen Konstruktor als eine Art Container vorstellen, der alle Eigenschaften und Methoden bei der Vereinbarung mit `new` an das neue Objekt übergibt. Im Gegensatz zu anderen Programmiersprachen, wo ein Konstruktor meist über eine Klasse definiert wird, benötigt man in JavaScript eine einfache Funktion → eine Konstruktorfunktion. Erst beim instanziieren mit `new` wird die normale Funktion zu einer objektorientierten Konstruktorfunktion.

```
function GiroKonto() { this.form = "Giro" }
var meinKonto = new GiroKonto;
```

Konstruktorfunktion: Im Beispiel wird der Funktion ein String-Wert (`besitzer`) übergeben die der Eigenschaft `.name` zugewiesen wird.

```
function meinSparbuch(besitzer){
   this.name = besitzer;
   this.kapital = 1200;

   this.meldung = function() {alert("Kontoinhaber: " + this.name);}
}
```

Vereinbart wird das neue Objekt (`sBuch1`) mit `new` und dem Namen der Konstruktorfunktion (`meinSparbuch`). Übergeben wurde der Besitzernamen als String. In Folge wird die Eigenschaft `.kapital` in der Console ausgegeben und die Methode `.meldung()` aufgerufen.

```
var sBuch1 = new meinSparbuch("Hans Berger");

console.log(sBuch1.kapital);
sBuch1.meldung();
```

Eine weitere Objekt-Vereinbarung ist problemlos möglich!

```
var sBuch2 = new meinSparbuch("Claudia Klein");
sBuch2.meldung();
```

Mit `.prototype.` lässt sich der Konstruktor erweitern! Diese Anweisungen sollten außerhalb der Konstruktorfunktion definiert werden. Hier um die Eigenschaften .sperre und .dauer.

```
meinSparbuch.prototype.art = "Prämiensparen";
meinSparbuch.prototype.dauer = 24;

var sBuch3 = new meinSparbuch();
alert("Art des Sparbuch: " + sBuch3.art);
```

Mit `.prototype` lässt sich sogar eine Konstruktorfunktion um eine zusätzliche erweitern. Man spricht dann von Vererbung:
`meinSparbuch.prototype = new GiroKonto;`