Übung 01.2

> Closures und Objekt Hierarchien

Closures

> Global variables can be made local (private) with closures.
> Closure is an inner function that has access to the outer (enclosing) function's variables - scope chain.
> Note that the inner function cannot call the outer function’s arguments object [...]
>
> *Richard. Understand JavaScript Closures With Ease. 2013. 01.03.2018.*
> *http://javascriptissexy.com/understand-javascript-closures-with-ease/*

```
var myStack = {
    data: [],
    count: 0,
    push: function(item) {
        this.data[this.count] = item;
        this.count++;
    },
    pop: function() {
        this.count--;
        var result = this.data[this.count];
        delete this.data[this.count];
        return result;
    }
}

console.log(myStack.pop()); // returns undefined

myStack.push(5);
myStack.push(8);
myStack.push(3);

console.log(myStack.pop()); // returns 3
```

Objekt Hierarchien

Erzeugen

```
function Shape(x, y) {
    this.x = x;
    this.y = y;
}

Shape.prototype.draw = function() {

}

var Circle    = new Shape(2,3);
var Rectangle = new Shape(2,3);

Circle.prototype.radius = 0;

Circle.prototype.setRadius = function(radius) {
    this.radius = radius;
}

Circle.setRadius(5);

Circle.prototype.draw = function() {
    return "Position(x:"+this.x+", y:"+this.y+"), Radius: "+this.radius;
}

Rectangle.prototype.width  = 0;
Rectangle.prototype.height = 0;

Rectangle.prototype.setWidth = function(width) {
    this.width = width;
}

Rectangle.prototype.setHeight = function(height) {
    this.height = height;
}

Rectangle.setWidth(2);
Rectangle.setHeight(3);

Rectangle.prototype.draw = function() {
    return "Position(x:"+this.x+", y:"+this.y+"), Measurements(width: "
            +this.width+", height: "+this.height+")";
}

// continued below...
```

Derived Properties

> Properties calculated out of other properties.

```
// ...continued here
Circle.prototype.area = function() {
    return Math.pow(this.radius,2)*Math.PI
}

Rectangle.prototype.area = function() {
    return this.width*this.height
}
```

Serialisierung

> Abbildung von strukturierten Daten auf eine sequenzielle Darstellungsform (`JSON.stringify(object)`)

```
var serializedCircle;
var serializedRectangle;
var parsedCircle;
var parsedRectangle;

Circle.toJSON = function(key) {
    var otherCircle = {
        x: this.x,
        y: this.y,
        radius: this.radius
    };
    return otherCircle;
}

Rectangle.toJSON = function(key) {
    var otherRectangle = {
        x: this.x,
        y: this.y,
        width: this.width,
        height: this.height
    };
    return otherRectangle;
}

serializedCircle    = JSON.stringify(Circle);
serializedRectangle = JSON.stringify(Rectangle);
parsedCircle        = JSON.parse(serializedCircle);
parsedRectangle     = JSON.parse(serializedRectangle;
```

Unterschiede zwischen Object-Hierarchien in JS und anderen OO-Sprachen

> More detailed info on MDN web docs:
> https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Details_of_the_Object_Model

OO JavaScript

- JS basiert auf Prototypen
- Hat nur Objekte (keine Klassen und Instanzen), wobei ein Objekt als Template für das nächste Objekt verwendet wird
- Jedes Objekt kann assoziiert werden als Prototype eines anderen Objekts (Properties werden vererbt)
- class chain

OO Andere Sprachen (Java, C++)

- Andere basieren auf Klassen und Instanzen
- Klassen definieren alle Properties, Instanzen instanzieren die Klassen
- Vererbung findet durch das Spezifizieren der Subklasse in der Subklasse selbst statt (Java: `public class Circle extends Shape {}`)
- property chain

Vor- und Nachteil

- Kommt immer auf die Anwendung an