Modules

Countdown Module V1.0

Da wir zuletzt soviel Zeit und Mühe in die Implementierung eines Countdowns investiert haben, möchten wir diesen ohne großen Aufwand wiederverwenden können. Hierfür werden wir ein *Modul* erstellen, das eine Countdown Funktion exportiert (also nach außen zur Verfügung stellt) und leicht in andere *Node.js* Apps eingebunden werden kann.

Lege ein neues WebStorm-Projekt `Countdown1` an und erstelle darin die JavaScript-Datei `countdown.js` an. Füge in letzterer den unten abgebildeten Code ein. Dieser sollte dir aus der letzten Übung bereits bekannt sein - bei eventuell vorhandenen Erinnerungslücken beschäftige dich bitte nochmals kurz mit der alten Aufgabe.

```
let counter = 10;

console.log(counter);

let timer = setInterval(() => {
    console.log(--counter)
}, 1000);

setTimeout(() => {
    clearInterval(timer)
}, counter * 1000 + 100);
```

Beim Ausführen der `countdown.js` erscheint die (hoffentlich) vertraute Ausgabe:

Passe jetzt die `countdown.js` so an, dass der Countdown per Funktionsaufruf gestartet werden kann. Als Parameter soll dabei der Startwert des Countdowns übergeben werden.

```
let setCountdown = function(counter)
{
    console.log(counter);

    let timer = setInterval(() => {
        console.log(--counter)
    }, 1000);

    setTimeout(() => {
        clearInterval(timer)
    }, counter * 1000 + 100);
}
```

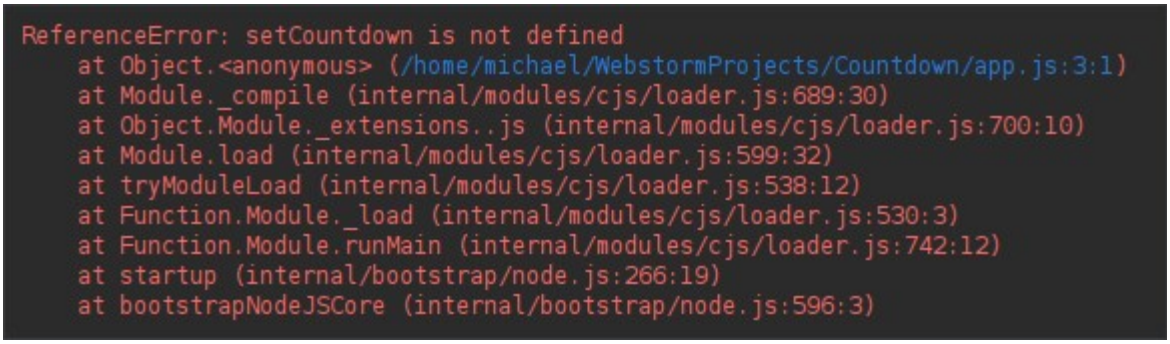
Rufe am Ende der `countdown.js` die obige Funktion mit dem Paramter *5* auf, also `setCountdown(5);` und starte die Applikation:

Als Nächstes wollen wir den Countdown in eine andere App einbinden. Erstelle eine neue JavaScript-Datei `app.js`, füge in diese den Aufruf `require("./countdown");` ein und führe die `app.js` aus. Wie du siehst wird wiederum der Countdown (mit Startwert *5*) auf der Konsole ausgegeben. Indem man den Pfad einer JavaScript-Datei an die Funktion `require` übergibt, wird nämlich der darin enthaltene Code ausgeführt.

Damit geben wir uns natürlich nicht zufrieden, sondern wollen in der `app.js` die Funktion `setCountdown` direkt aufrufen können. Somit wären wir beispielsweise in der Lage, andere Startwerte festzulegen oder weitere Countdowns zu starten. Lösche zunächst den Aufruf `setCountdown(5);` aus der `countdown.js`. Füge anschließend in die `app.js` einen weiteren Aufruf ein, sodass die JavaScript-Datei nachher folgendermaßen aussieht.

```
require("./countdown");

setCountdown(10);
```

Beim Ausführen der `app.js` erhältst du nun jedoch eine Fehlermeldung. Diese besagt, dass die Funktion `setCountdown` in der `app.js` nicht bekannt ist.



Wie bereits in der Einleitung angekündigt, muss ein Modul jene Funktionen, die von außen aufgerufen werden dürfen, zuvor **exportieren**. Füge daher am Ende der `countdown.js` die Anweisung `module.exports = setCountdown;` hinzu. `module.exports` legt den Rückgabewert der `require` Funktion fest. Dieser muss dann in der `app.js` nur noch einer entsprechend benannten Variable zugewiesen werden, und schon ist die Funktion `setCountdown` verfügbar.

```
const setCountdown = require("./countdown");

setCountdown(10);
```

Der Countdown sollte sich jetzt problemlos ausführen lassen. In obigem Code haben wir den Rückgabewert von `require` einer Konstanten zugewiesen. Dies wird empfohlen, um ein irrtümliches Überschreiben der Funktion zu verhindern.

Countdown Module V2.0

Wir möchten unser Modul weiterentwicklen und noch eine weitere Funktion exportieren. Mit deren Hilfe soll es möglich sein, die Geschwindigkeit des Countdowns zu verändern. Lege hierfür ein neues WebStorm-Projekt `Countdown2` an und erstelle darin die JavaScript-Dateien `app.js` und `countdown.js` . Füge darin den Code aus dem vorherigen Projekt mithilfe von *Copy & Paste* ein.

Da wir in weiterer Folge mehrere Funktionen exportieren möchten, weisen wir `module.exports` (im Gegensatz zum letzten Projekt) kein anderes Objekt zu. Stattdessen verändern wir das vorhandene Objekt, indem wir die gewünschten Methoden hinzufügen. Ändere hierfür die Export-Anweisung in der `countdown.js` folgendermaßen:

```
module.exports.setCountdown = setCountdown;
```

Auch die `app.js` muss noch etwas angepasst werden. Da der Rückgabewert von `require` nicht mehr der Funktion entspricht, passen wir den Namen der Konstante an. Dabei ist es üblich, einfach den Namen des Moduls (in diesem Fall also `countdown`) zu verwenden. Die Funktion kann dann wie

unten dargestellt aufgerufen werden.

```
const countdown = require("./countdown");

countdown.setCountdown(10);
```

Um die Geschwindigkeit des Countdowns beinflussen zu können, deklarieren wir am Beginn der `countdown.js` eine Variable die festlegt, wie oft der Zähler pro Sekunde erniedrigt wird. Deren Wert muss dann natürlich beim Aufruf von `setInterval` und `setTimeout` berücksichtigt werden. Passe die `countdown.js` entsprechend an - sie sollte nachher wie folgt aussehen:

```
let decrementsPerSecond = 1;

let setCountdown = function(counter)
{
    console.log(counter);

    let timer = setInterval(() => {
        console.log(--counter)
    }, 1000 / decrementsPerSecond);

    setTimeout(() => {
        clearInterval(timer)
    }, counter * 1000 / decrementsPerSecond + 100);
}

module.exports.setCountdown = setCountdown;
```

Führe die Applikation aus - die Geschwindigkeit sollte derzeit noch unverändert sein, da für `decrementsPerSeconds` der Initialwert *1* gewählt wurde. Füge in der `countdown.js` als Nächstes die folgende Funktion und Export-Anweisung hinzu, um die Geschwindigkeit von außen verändern zu können.

```
let setDecrementsPerSecond = function(decrements){
    decrementsPerSecond = decrements;
}

module.exports.setDecrementsPerSecond = setDecrementsPerSecond;
```

Füge zuletzt in der `app.js` vor dem Aufruf von `setCountdown` noch die Anweisung `countdown.setDecrementsPerSecond(3);` ein. Beim Ausführen der App zählt der Countdown jetzt dreimal so schnell herunter als wie zuvor.

Countdown Module V3.0

Als letzte Erweiterung des Moduls wollen wir dem Countdown nun auch noch eine *callback function* übergeben, die dieser am Ende aufruft. Erstelle hierfür ein neues WebStorm-Projekt `Countdown3` und lege darin die JavaScript-Dateien `app.js` und `countdown.js` . Füge darin den Code aus dem vorherigen Projekt mithilfe von *Copy & Paste* ein.

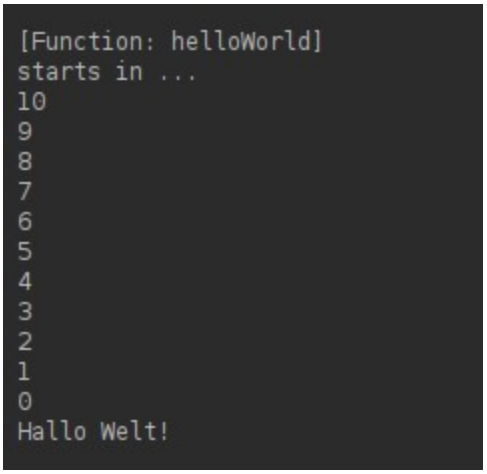
In der `countdown.js` erweitern wir die Funktion `setCountdown` noch um einen `callback` Parameter. Füge zu Beginn der Funktion zwei Konsolenausgaben mit zusätzlichen Informationen für den Benutzer hinzu. Die übergebene *callback function* muss anschließend in der *arrow function expression* von `setTimeout` aufgerufen werden. Nachdem du alle Änderungen durchgeführt hast, sollte `setCountdown` folgendermaßen aussehen:

```
let setCountdown = function(callback, counter)
{
    console.log(callback);
    console.log("starts in ...")

    console.log(counter);

    let timer = setInterval(() => {
        console.log(--counter)
    }, 1000 / decrementsPerSecond);

    setTimeout(() => {
        clearInterval(timer);
        callback();
    }, counter * 1000 / decrementsPerSecond + 100);
}
```

Ergänze als Nächstes die `app.js` um diese Funktion:

```
var helloWorld = function(){
    console.log("Hallo Welt!")
}
```

Gib schließlich `helloWorld` , beim Aufruf von `setCountdown` , als *callback function* an, also `countdown.setCountdown(helloWorld, 10);` . Wenn du die App anschließend startest, sollte die Konsolenausgabe wie unten abgebildet aussehen:



Operations Module

Bei der letzte Aufgabe soll nun das Exportieren von Funktionen nochmals selbstständig geübt werden. Lege ein neues WebStorm-Projekt `BasicOperations` an und erstelle darin die JavaScript-Datei `operations.js` an. Füge in letzterer den unten abgebildeten Code ein.

```
function sum(num1, num2){
    return num1 + num2;
}

function difference(num1, num2){
    return num1 - num2;
}

function product(num1, num2){
    return num1 * num2;
}

function quotient(num1, num2){
    return num1 / num2;
}
```

Die oben dargestellten Funktionen für die vier Grundrechnungsarten sollten dir aus der Übungsaufgabe *SimpleCalculator* bereits bekannt sein. Erstelle als Nächstes die Datei `app.js` und implementiere dort die folgende Funktion. Ersetze innerhalb von `printCalculation` die Zahl 50 durch deine **Katalognummer**.

```
// Ersetze 50 durch deine Katalognummer
function printCalculation(operator, calculate){
    console.log("50 " + operator + " 7 = " + calculate(50, 7));
```

```
}
```

Erledige jetzt folgende Aufgaben:

- Exportiere alle Funktionen des Moduls `operations.js` .
- Binde dieses Modul in der `app.js` ein.
- Gib in der `app.js` die vier Grundrechnungsarten (mithilfe der importierten Funktionen und `printCalculation`) aus.

Lautet deine Katalognummer also beispielsweise *50*, so sollte die Konsolenausgabe der App folgendermaßen aussehen:

```
50 + 7 = 57
50 - 7 = 43
50 * 7 = 350
50 / 7 = 7.142857142857143
```