Async Functions

Clock

In dieser Übung implementieren wir als Erstes eine Uhr, die abwechselnd "tick" und "tack" ausgibt. Erstelle ein neues WebStorm Projekt Clock und lege darin die JavaScript-Datei clock.js an.

Für den Anfang begnügen wir uns damit, dass einmal pro Sekunde "tick" auf der Konsole erscheint. Programmiere hierfür die folgende Funktion:

```
let tickTack = function(){
    console.log("tick");
}
```

Mithilfe von setInterval können wir tickTack einmal pro Sekunde (also einmal pro 1000 Millisekunden) aufrufen.

```
let timer = setInterval(tick, 1000);
```

Die Variable timer werden wir später benötigen, um die Uhr auch wieder ausschalten zu können. Führe das Programm aus und prüfe ob du folgende Konsolenausgabe erhältst:

```
tick
tick
tick
tick
tick
tick
```

Damit die Uhr auch "tack" ausgibt, deklarieren wir oberhalb der Funktion die Variable logTick. In der Function tickTack entscheiden wir dann aufgrund des Werts dieser boolschen Variable, ob "tick" oder "tack" ausgegeben wird.

```
let logTick = true;
let tickTack = function(){
   if(logTick){
      console.log("tick");
   } else {
      console.log("tack");
```

```
}
```

Wenn du das Programm ausführst, stellst du fest, dass nachwievor nur "tick" ausgegeben wird. Um die abwechselnde Ausgabe zu erreichen, müssen wir am Ende von ticktack den Wert von logTick auch noch ändern.

```
let tickTack = function(){
    if(logTick){
        console.log("tick");
    } else {
        console.log("tack");
    }

    logTick = !logTick;
}
```

Wenn du das Programm nun erneut ausführst, solltest du folgende Ausgabe erhalten:

```
tick
tack
tick
tack
tick
tack
tick
```

Alarm

Jetzt möchten wir unsere Uhr auch als Wecker verwenden, der uns nach sechs Sekunden aufweckt (ein Powernap;-)) und sich automatisch abschaltet (also kein lästiges "tack tack" mehr von sich gibt). Erstelle im Calculator Projekt eine neue JavaScript-Datei alarmApp.js. Füge darin den gesamten Inhalt der clockApp.js mithilfe von Copy & Paste ein.

Tätige am Ende der alarmApp.js folgenden Funktionsaufruf, um den Wecker nach sechs Sekunden wieder auszuschalten:

```
setTimeout(function(){
    clearInterval(timer);
}, 6000);
```

Beim Ausführen des Programms wirst du feststellen, dass das letzte "tack" nicht ausgegeben wird. Ändere daher den Zeitparameter von setTimeout auf 6100 und führe das Programm nochmals aus. Jetzt sollte die Ausgabe stimmen.

```
tick
tack
tick
tack
tick
tack
```

Als Nächstes erweitern wir unseren Wecker um eine Sprachfunktion - er soll uns nach dem letzten "tack" mit einem fröhlichen "Guten Morgen" begrüßen. Füge hierfür ganz am Ende der alarmApp.js den Aufruf console.log("Guten Morgen"); ein und starte die Applikation.

```
Guten Morgen
tick
tack
tick
tack
tick
tack
```

Wie oben dargestellt, wird die Begrüßung bereits vor dem ersten "tick" ausgegeben. Da wir die Funktion tickTack asynchron aufgerufen haben, wird der restliche Code der alarmApp.js zuvor noch ausgeführt. Nachdem die Konsolenausgabe am Ende der alarmApp.js also offensichtlich falsch ist, kannst du sie an dieser Stelle wieder löschen. Überlege selbstständig wo du console.log("Guten Morgen"); richtigerweise einfügen musst, damit die App folgendes ausgibt:

```
tick
tack
tick
tack
tick
tack
Guten Morgen
```

Advanced Alarm

Da ein Wecker, der uns immer nach sechs Sekunden weckt, zugegebenermaßen ziemlich unbrauchbar ist, müssen wir ihn wohl noch etwas verbessern. Erstelle im Calculator Projekt eine neue JavaScript-Datei advancedAlarmApp.js. Füge darin den gesamten Inhalt der alarmApp.js mithilfe von Copy & Paste ein.

Zuerst sorgen wir dafür, dass unsere Implementierung noch etwas kompakter wird. Da wir die Funktion tickTack nur ein einziges Mal verwenden, können wir sie beim Aufruf von setInterval einfach als anonyme Funktion übergeben. Dabei verwenden wir natürlich wieder eine arrow function expression. Ändere den Aufruf von setInterval im advancedApp.js also wie folgt:

```
let timer = setInterval(() => {
    if(logTick){
        console.log("tick");
    } else {
        console.log("tack");
    }
    logTick = !logTick;
}, 1000);
```

Da die Funktion tickTack jetzt nicht mehr aufgerufen wird, solltest du sie löschen. Verwende beim Aufruf von setTimeout ebenfalls eine arrow function expression. Teste anschließend ob die Ausgabe deines Programm unverändert ist.

Jetzt wollen wir aber unsere Schlafphasen endlich etwas verlängern und deklarieren hierfür am Anfang advancedApp.js folgende Variable:

```
let seconds = 10;
```

Ändere anschließend den Zeitparamter von setTimeout auf seconds * 1000 + 100 und führe das Programm nochmals aus. Jetzt sollte dich der Wecker erst nach zehn Sekunden wecken - zugegebenermaßen immer noch nicht sehr erholsam. Teste dein Programm daher auch mit anderen Werten für die Variable seconds.

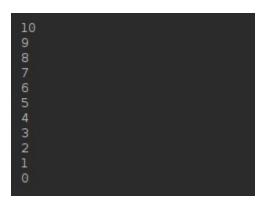
Countdown

Als letzte Übung entwicklen wir jetzt noch eine Countdown-App. Erstelle im Calculator Projekt eine neue JavaScript-Datei countdown.js. Füge darin den gesamten Inhalt der advancedAlarmApp.js mithilfe von Copy & Paste ein.

Benenne im countdown.js die Variable seconds (mithilfe der Tastenkombination Umschalt + F6) in counter um. Gib gleich darunter den Initialwert der Variable aus.

```
let counter = 10;
console.log(counter);
```

Überlege nun selbstständig, wie du den Code der arrow function expression beim Aufruf von setIntervall verändern musst, damit die Konsolenausgabe wie unten dargestellt aussieht. Ein kleiner Tipp: Die Variable counter muss jeweils um die Zahl Eins erniedrigt werden.



Wenn du es geschafft hast: Herzlichen Glückwunsch! Teste deinen Countdown abschließend noch mit anderen Startwerten.