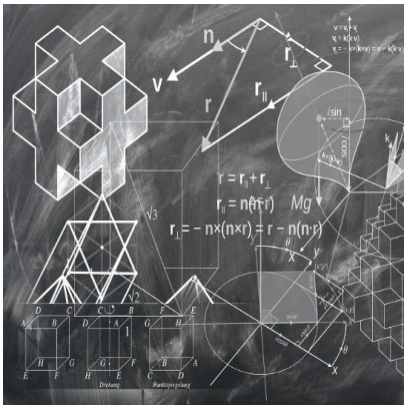

Interaktives Formular mit JavaScript

Description

In der Webentwicklung brauchen wir oft Formulare. Die meisten funktionieren sehr simpel, doch manchmal will der Benutzer schon bei der Eingabe ein Ergebnis sehen. Hierfür ist JavaScript nahezu perfekt. Im heutigen Beispiel gehen wir es etwas komplexer an und machen uns ein Formular, in dem Werte umgerechnet werden.

Was wir vorhaben, findest Du hier. Wir rechnen Längenmaße und Temperaturen um. Natürlich kannst Du dies um Flächen, Volumen und Massen erweitern oder ganz andere Dinge damit machen.

HTML Grundgerüst

Wie eigentlich immer, starten wir auch dieses Mal mit einem HTML-Grundgerüst.

```
<!DOCTYPE html>
<html lang="de">
 <head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <script src="js/umrechnung.js"></script>
  <title>Umrechnung</title>
        <body>
 </body>
</html>
```

In Zeile 6 sehen wir, dass ein JavaScript mit dem Dateinamen *umrechnung.js* im Ordner *js* geladen wird. Das kannst Du gleich anlegen. Der Rest des HTMLs kommt zwischen die Body-Tags.

Formular

Wir erstellen nun zwei Formulare, jeweils mit einer <*h3*> Überschrift davor. Die Formulare schließen eine Tabelle ein. Man kann es auch ohne Tabelle machen, aber so lässt es sich recht geschickt

formatieren. Auf CSS und entsprechende Klassen habe ich dieses Mal verzichtet.

```
<h3>Längen</h3>
<div>
 <form id="langen" name="langen">
  <table>
   <tr>
    <td>
     <input oninput=calcLengthen() type="number" name="l1" id="l1" step="0.001
     <select name="lengthen1" id="lengthen1" onchange=calcLengthen() tabindex=
      <option value="le1">Kilometer (km)</option>
      <option value="le2">Meter (m)</option>
      <option value="le3" selected>Dezimeter (dm)</option>
      <option value="le4">Zentimeter (cm)</option>
      <option value="le5">Millimeter (mm)</option>
      <option value="le6">Mikrometer (µm)</option>
     </select>
    </td>
   <tr>
   </tr>
    <td>
     <input type="text" name="l2" id="l2" value=0 readonly>
     <select name="lengthen2" id="lengthen2" onchange=calcLengthen() tabindex=
      <option value="le1">Kilometer (km)</option>
      <option value="le2">Meter (m)</option>
      <option value="le3" selected>Dezimeter (dm)</option>
      <option value="le4">Zentimeter (cm)</option>
      <option value="le5">Millimeter (mm)</option>
      <option value="le6">Mikrometer (µm)</option>
     </select>
    </td>
   </tr>
  </table>
 </form>
</div>
<h3>Temperaturen</h3>
<div>
 <form id="temperatures" name="temperatures">
  <table>
   <tr>
    <td>
     <input oninput=calcTemperatures() type="number" name="t1" id="t1" step="0
     <select name="temp1" id="temp1" onchange=calcTemperatures() tabindex="5">
      <option value="te1" selected>Grad Celsius (°C)</option>
      <option value="te2">Grad Fahrenheit (°F)</option>
      <option value="te3">Kelvin (K)</option>
     </select>
    </td>
   <tr>
   </tr>
    <td>
     <input type="text" name="t2" id="t2" value=0 readonly>
     <select name="temp2" id="temp2" onchange=calcTemperatures() tabindex="6">
      <option value="te1" selected>Grad Celsius (°C)</option>
      <option value="te2">Grad Fahrenheit (°F)</option>
```

```
      <option value="te3">Kelvin (K)</option>
     </select>
    </td>
   </tr>
  </table>
 </form>
</div>
```

Wichtig sind zunächst die IDs der Formulare und Inputfelder.

Was passiert hier?

Wir haben zwei Formulare, die identisch funktionieren. Deshalb schauen wir uns nur das erste genauer an.

```
<form id="langen" name="langen">
```

Hier beginnt das Formular mit der ID *lange*.

Eingabefeld

```
<input oninput=calcLengthen() type="number" name="l1" id="l1" step="0.001"  pl
```

Wir haben ein Eingabefeld vom Typ *number*. Das ist ganz wichtig, denn dadurch kann man in vorgegebenen Schritten (*steps*) hoch oder runter gehen. Außerdem erhält man auf dem Smartphone ein hübsches Tastenfeld, wenn man es anklickt. Besonders wichtig ist *oninput=calcLengthen()*. Das bedeutet, dass immer, wenn etwas eingegeben wird, das Script *calcLengthen()* gestartet wird. Als Platzhalter und Titel nennen wir es „Ausgangswert”, die ID lautet *l1*.

Tabindex bestimmt die Reihenfolge, wohin die Selektion springt, sobald die Tabulatortaste gedrückt wird. Ich halte das vor allem bei großen Formularen für wichtig.

Dimensionsauswahl

```
<select name="lengthen1" id="lengthen1" onchange=calcLengthen() tabindex="2">
 <option value="le1">Kilometer (km)</option>
 <option value="le2">Meter (m)</option>
 <option value="le3" selected>Dezimeter (dm)</option>
 <option value="le4">Zentimeter (cm)</option>
 <option value="le5">Millimeter (mm)</option>
 <option value="le6">Mikrometer (µm)</option>
</select>
```

Bei den Längen haben wir sechs Auswahlmöglichkeiten. Wir könne sogar Kilometer in Mikrometer umwandeln. Wichtig ist hier *onchange=calcLengthen()*. Außerdem steht bei Dezimeter *selected* als Vorauswahl.

Das Gute am Formular ist, dass wir keine Buttons brauchen. Alles passiert während der Benutzung.

JavaScript Vorwissen

Wenn Du mit dem oben verlinkten Beispiel ein wenig herumgespielt hast, sind Dir vielleicht drei Dinge aufgefallen.

1. Es wird erst ein Ergebnis angezeigt, wenn eine Zahl im Feld eingegeben wurde.
2. Größere Zahlen werden formatiert.
3. Bei hohen Stellen wird das Resultat ungenau.

1 und 2 sind Features, 3 ist ein Problem.

JavaScript rechnet falsch

JavaScript ist eine praktische Programmiersprache, aber mit Mathe hat es ein Problem. Das geht aber nicht nur JavaScript so. Alle Sprachen, die Fließkommazahlen nach IEEE 754 implementieren, weisen Rundungsfehler auf. Wir verwenden den Datentyp Number. Das ist ein 64-Bit-Floating-Point-Datentyp. Jede Zahl wird in einer binären Exponential-Repräsentation abgebildet. Für Ganzzahl-Werte bedeutet dies, dass eine gesicherte Darstellung bis 15 Ziffern möglich ist, bevor Rundungsfehler auftreten. Um das Problem zu beheben, gibt es im Internet einige Bibliotheken, auf die ich bei diesem Beispiel bewusst verzichte, um es einfach zu halten. Es reicht an dieser Stelle, wenn Du weißt, dass es dieses Problem gibt.

Zahl Formatieren

Da wir später auf diese Funktionen zugreifen, stelle ich zunächst die „Hilfsfunktionen” vor. Eine kleine Funktion ist für die Formatierung der Zahl zuständig. In einem Formular kann es sehr oft auftreten, weshalb wir es auslagern. Außerdem kann es sein, dass wir irgendwann die Formatierung ändern. Dann haben wir nur eine Anlaufstelle.

```
function numFormatter(num)
{
 return num.toLocaleString(undefined,{maximumFractionDigits: 20});
}
```

Im Prinzip besteht die Funktion nur aus einer Zeile. Die Methode *toLocaleString()* gibt eine Zeichenkette mit einer sprachabhängigen Darstellung zurück. Mit den neuen Argumenten *locales* und *options* können Anwendungen die Sprache angeben, deren Formatierungskonventionen verwendet werden sollen, und das Verhalten der Funktion anpassen. Was die Zeile macht, ist die Formatierung der Zahlen auf 20 Stellen, indem es Punkte zur besseren Lesbarkeit einfügt. Ursprünglich gedacht war die Methode zur Formatierung von Zeitangaben.

Die Funktion kann man natürlich umschreiben. Etwa, dass es bei Tausendern die Zahl abschneidet und hinten „k” steht. Also *2k* statt *2000*.

Funktion der Umwandlung

Hier gibt es mehrere Möglichkeiten. Prinzipiell kann man diese Funktion auch in der Hauptfunktion (weiter unten) unterbringen. Ich habe mich aber für eine Auslagerung entschieden, weil mir die Handhabung, wenn man es bspw. um Flächen und Volumen ergänzt, einfacher erschien.

Das Logische Problem besteht darin, dass wir sechs Längenangaben haben, die man frei kombinieren kann. Das macht 30 Kombinationen, wenn man die Gleichen nicht berücksichtigt. Zwischen zwei Schritten gibt es aber immer einen Faktor, weshalb ich mich für folgende Funktion entschied:

```
function dimensionLengthen(dim)
{
 let fac;
 switch (dim)
 {
  // km
  case „le1":
   fac = 1000000;
  break;
  // m
  case „le2":
   fac = 1000;
  break;
  // dm
  case „le3":
   fac = 100;
  break;
  // cm
  case „le4":
   fac = 10;
  break;
  // mm
  case „le5":
   fac = 1;
  break;
  // µm
  case „le6":
   fac = 0.001;
  break;
  default:
   fac = 1;
  break;
 }

 return fac;
}
```

Wirkt auf den ersten Blick komisch. Den Startpunkt habe ich bei Millimeter angesetzt. Von mm auf cm ist es 10, von mm auf dm 100 usw. Logischer erscheint auf den ersten Blick, Mikrometer als 1 zu nehmen, aber wenn wir es mit Flächen oder gar Volumen zu tun haben, laufen wir viel eher in das o. g. Rundungsproblem. Was hier aber genau passiert, verstehen wir besser, wenn wir die nächste Funktion

anschauen.

Längenmaße umrechnen

```
function calcLengthen()
{
 let vl1 = parseFloat(document.getElementById("l1").value);

 if (!isNaN(vl1))
 {
  let dimension1 = document.getElementById("lengthen1").value;
  let dimension2 = document.getElementById("lengthen2").value;
  let fac1, fac2, result;

  fac1 = dimensionLengthen(dimension1);
  fac2 = dimensionLengthen(dimension2);

  result = vl1 / fac2 * fac1;
  document.getElementById("l2").value = numFormatter(result);
 }
}
```

Das ist die Funktion, die wir im Formular aufrufen.

Damit holen wir uns den Wert aus dem Eingabefeld. Die Funktion *parseFloat()*
 parst ein Argument (und konvertiert es bei Bedarf zuerst in einen String) und
parseFloat
 parst Nicht-String-Objekte, wenn es eine toString- oder valueOf-Methode hat.
parseFloat auf dem Ergebnis dieser Methoden aufgerufen worden wäre.

```
if (!isNaN(vl1))
```

Das ist eines der Features. Wir schauen, ob das Feld leer ist. Die folgenden A
isNaN()
 ermittelt, ob ein Wert NaN ist oder nicht. Ob es das ist, erfahren wir von
parseFloat.

Doch wann kommt das überhaupt vor? Schließlich haben wir den Typ *number*
 und rufen die Funktion nur auf, wenn eine Zahl eingegeben wird. Wirklich? Nei
NaN angezeigt. Das steht für „Not-A-Number” und sieht sehr unschön aus.

Wir schauen, welche Längen gewählt wurden und deklarieren außerdem die Variabl
fac1, *fac2* und *result*. Zur Erinnerung: Die *let*
-Anweisung deklariert eine lokale Variable im Blockbereich und initialisiert s

```
fac1 = dimensionLengthen(dimension1);
fac2 = dimensionLengthen(dimension2);
```

Nun holen wir uns unsere Faktoren aus der Funktion *dimensionLengthen*. Da wir sie zweimal aufrufen, hat sich die Auslagerung bereits gelohnt.

```
document.getElementById("l2").value = numFormatter(result);
```

Wir schreiben das Ergebnis in das Feld mit der ID *l2*.

Beispiel

Noch einmal im Detail an einem konkreten Beispiel, um zu verstehen, was passie

Nehmen wir an, der User gibt als Wert 12 ein und will Meter in Millimeter umwa

Faktor 1 ist Meter. *dimension1* ist im Auswahlfeld „le2?. Mit *fac1 = dimensionLengthen(dimension1)* wollen wir wissen, was für einen Faktor „le2? nun hat. Die Funktion *dimensionLengthen* wird aufgerufen und die Switch gibt aus, dass „le2? 1000 ist. Also bekommt *fac1* den Wert 1000 zugewiesen. Das Gleiche passiert mit *fac2*. *dimension2* beinhaltet „le5? und „le5? ist 1. Somit haben wir unsere beiden Faktoren, näm

Das Resultat errechnen sich nun wie folgt: 12 / 1 * 1000. Wir erhalten als Erg

Temperaturen berechnen

Bei Temperaturen ist es nicht ganz so einfach, weil die Berechnungen immer and

```
function calcTemperatures()
{
 let te1 = parseFloat(document.getElementById("t1").value);

 if (!isNaN(te1))
 {
  let dimension1 = document.getElementById("temp1").value;
  let dimension2 = document.getElementById("temp2").value;
```

```
  const kelvin = 273.15;
  let result;

  if (dimension1 === dimension2)
  {
   result = te1;
  } else {
   // °C zu K
   if (dimension1 === „te1" && dimension2 === „te3")
   {
    result = te1 + kelvin;
   }

   // K zu °C
   if (dimension1 == „te3" && dimension2 == „te1")
   {
    result = te1 - kelvin;
   }

   // °C zu Farenheit
   if (dimension1 === „te1" && dimension2 === „te2")
   {
    result = 1.8*te1+32;
   }

   // Farenheit zu °C
   if (dimension1 === „te2" && dimension2 === „te1")
   {
    result = (te1-32)/1.8;
   }

   // K zu Farenheit
   if (dimension1 === „te3" && dimension2 === „te2")
   {
    result = (te1-kelvin)*1.8+32;
   }

   // Farenheit zu K
   if (dimension1 === „te2" && dimension2 === „te3")
   {
    result = (te1-32)*(5/9)+kelvin;
   }
  }

  document.getElementById("t2").value = numFormatter(result);
 }
}
```

Im Kern geht es so, wie bei den oberen Funktionen, nur haben wir statt einer *Switch* einzelne *if*s. Da wir *kelvin* mehrmals brauchen, wurde es als Konstante definiert. Und eine Besonderheit gi

Berechnung von Ligen bzw. einem Ligaturnier. Die Formel lautet x^2
-x. So ergeben sich sechs Kombinationen bei den Temperaturen, wenn man zwei Au

Weiterführende Links

Webentwicklung Grundkurs Teil 1

Passwortgenerator in PHP

Die Zukunft der Spieleentwicklung

Warum soll ich programmieren lernen?

Date Created
30. Juli 2021
Author
sven