Tutorial 1 – DOM

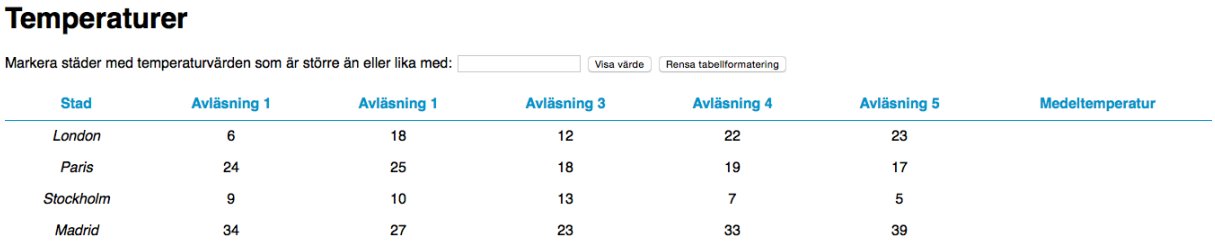
Syfte

I denna tutorial kommer du att lära dig att

1. Avläsa olika elementnoder i DOM-trädet inklusive textnoder och attributnoder.
2. Göra smärre beräkningar.
3. Skapa textnoder med resultat och placera in dem i element i DOM-trädet.
4. Förändra sidans utseende med CSS.

Bakgrund

Dokumentet innehåller temperaturvärden för ett antal huvudstäder. Dessa visas i en tabell. För varje stad visas fem temperaturvärden. I bilden visas dokumentets utseende när JavaScript är avslaget. Observera att kolumnen Medeltemperatur är tom.



Figur 1: Sidan med JavaScript avslaget.

Din uppgift

Med utgångspunkt från tabellens data, skall din lösning göra följande:

1. Beräkna medeltemperatur för varje stad samt sätta in dessa medelvärden som textnoder i tabellen.
2. Skapa en funktionen som tar ett temperaturvärde som användaren skriver in inmatningsfältet ovanför tabellen. Funktionen skall leta fram alla temperaturvärden som är lika stora eller större än detta inmatade värde och ändra class-attribut (som är kopplat till CSS) så att dessa värden (och raden/raderna som innehåller värdet/värden) visas på ett tydligt sätt.
3. Skapa en funktion som har till uppgift att rensa bort formateringen som utförs av funktionen i punkt två ovan.

Filer som ingår i lösningen

1. temperatur.html – denna fil skall inte förändras.
2. temperatur.css – denna fil skall inte förändras.
3. temperatur.js – *denna fil skall du skapa.*

Filerna placeras i samma mapp så att sökvägen till filerna stämmer.

HTML-dokumentet temperatur.html

```
<!DOCTYPE html>
<html lang="sv">
<head>
   <meta charset="utf-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <title>Temperaturer</title>
   <link rel="stylesheet" type="text/css" href="temperatur.css">
   <script type="text/javascript" src="temperatur.js"></script>
</head>
<body>
   <h1>Temperaturer</h1>
   <form>
      <label>Markera städer med temperaturvärden som är större än eller lika med: <input
id="varde" type="text" /></label>
      <input type="button" onclick="hitta();" value="Visa värde"/>
      <input type="button" onclick="rensa();" value="Rensa tabellformatering" />
   </form>


   <table id="temperaturtabell">
      <thead>
         <th>Stad</th>
         <th>Avläsning 1</th>
         <th>Avläsning 1</th>
         <th>Avläsning 3</th>
         <th>Avläsning 4</th>
         <th>Avläsning 5</th>
         <th>Medeltemperatur</th>
      </thead>
      <tbody>
         <tr>
            <td class="stad">London</td>
            <td>6</td>
            <td>18</td>
            <td>12</td>
            <td>22</td>
            <td>23</td>
            <td class="medel"></td>
         </tr>
         <tr>
            <td class="stad">Paris</td>
            <td>24</td>
            <td>25</td>
            <td>18</td>
            <td>19</td>
            <td>17</td>
            <td class="medel"></td>
         </tr>

         <tr>
            <td class="stad">Stockholm</td>
            <td>9</td>
            <td>10</td>
            <td>13</td>
            <td>7</td>
            <td>5</td>
            <td class="medel"></td>
         </tr>
         <tr>
            <td class="stad">Madrid</td>
            <td>34</td>
            <td>27</td>
            <td>23</td>
            <td>33</td>
            <td>39</td>
```

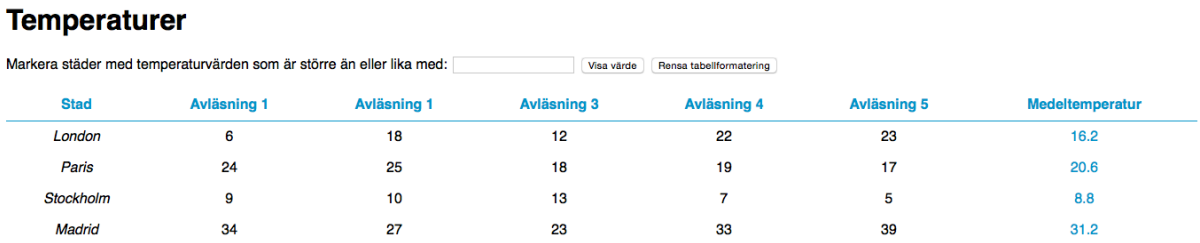
```
            <td class="medel"></td>
          </tr>
        </tbody>
    </table>
</body>
</html>
```

CSS-dokumentet temperatur.css

```
body {
    font-family: Helvetica, Arial, sans-serif;
    margin: 3em;
}

table {
    margin-top: 1em;
    border-collapse: collapse;
    width: 100%;
    color: inherit;
    line-height: 1.2em;
}

th, td {
    padding: 0.5em;
    text-align: center;
}

th {
    color: #0099CC;
    border-bottom: 1px solid #0099CC;
}

td.stad {
    font-style: italic;
}

td.medel {
    color: #0099cc;
}

tr.emp td {
    background: rgb(255, 223, 213);
}

td.emptd {
    font-weight: bold;
    color: #ff3300;
}
```

Del 1 – Beräkning och visning av medeltemperatur

Tabellen har sju kolumner. De sex första (staden och temperaturvärdena) innehåller data medan den sista är tom. Vi skall använda oss av JavaScript för att räkna ut medeltemperatur för varje stad genom att skapa en ny textnod för varje medelvärde och sätta in denna nod i td-elementen för den aktuella kolumnen. Resultatet skall se ut på följande sätt:



Stad	Avläsning 1	Avläsning 1	Avläsning 3	Avläsning 4	Avläsning 5	Medeltemperatur
London	6	18	12	22	23	16.2
Paris	24	25	18	19	17	20.6
Stockholm	9	10	13	7	5	8.8
Madrid	34	27	23	33	39	31.2

Figur 2: Medelvärde uträknat och infogat m.h.a. JavaScript.

Nu ska vi börja arbetet med att skapa en ny fil, *temperatur.js*.

Funktion raknaUtMedelvarde

Medelvärdet skall vi räkna ut i en funktion som skall heta raknaUtMedelvarde. Denna funktion skall startas så fort som sidan är inläst i webbläsaren. Därför börjar vi med att se till att detta funktionsanrop utförs.

```
window.onload = raknaUtMedelvarde;
```

Detta betyder att metoden raknaUtMedelvarde skall anropas när händelsen onload uppstår; när sidan är inläst. Nu skall vi skapa själva funktionen *raknaUtMedelvarde* (den placeras under window.onload):

```
function raknaUtMedelvarde()
{
}
```

Vi vill hämta alla tabellrader (tr-element). Dessa skulle kunna hämtas direkt då de är de enda tr-element som finns i dokumentet. Trots det så börjar vi med att ta fram tabellen först. Anledningen är att visa hur man kan identifiera rätt rader om det skulle finnas mer än en tabell i dokumentet. Tabellen har id *"temperaturtabell"*, och med hjälp av metoden *getElementsById* kan vi hämta ut ett givet element ur DOM-trädet på basis av dess id. Referensen till tabellen lagras i en variable kallad *tabell*.

```
var tabell = document.getElementById("temperaturtabell");
```

Nu är tabellen identifierad och vi kan gå vidare med att hämta fram tabellens rader (tr-element) genom att använda *getElementsByTagName* på variabeln (objektet) *tabell*. Fanns det flera tabeller i dokumentet så är det viktigt att veta att vi hämtar rader från rätt tabell. Referensen till raderna sparas i variabeln *rader*. Denna variabel är en array: den kan peka på noll, en eller flera rader. Även om den bara refererar till en rad så är det en array. En array är en samling data av en viss typ.

```
var rader = tabell.getElementsByTagName("tr");
```

Vi skall nu gå igenom var och en av dessa rader. Även om vi vet att det är fyra rader så skall vi skapa en lösning som fungerar för noll, en eller ett okänt antal rader. Det uppnår vi genom att iterera (upprepa) arbetet m.h.a. en loop. För vårt ändamål är en for-loop lämplig. Vår loop skall iterera över alla rader. Antalet rader får vi fram genom att kontrollera längden på array:en *rader*, m.h.a. *rader.length*. I nedan kod använder vi variabeln *i* som räknare. Variabeln räknas upp för varje varv, och vi kan indexera (peka ut aktuell rad) med hjälp av denna variabel.

```
for(var i = 1; i < rader.length; i++){
//här kommer vi att fylla på med kod.
}
```

Det första vi gör i loopen är att ta fram alla tabellceller (td-element) för den aktuella raden. Vi använder *getElementsByTagName* på *rader[i]* vilket är aktuell rad. Det första varvet är värdet på *i* = 1 för att skippa tabellhuvudet och gå direkt till den första raden som har numerisk data. Variabeln *tabellceller* pekar på de td-element som vi har hämtat i nedanstående kod. Observera att även tabellcellerna lagras i en array.

```
var tabellceller = rader[i].getElementsByTagName("td");
```

Nu skall vi gå igenom alla tabellcellerna för den aktuella raden. Vi skulle kunna göra detta i loopen, men för att få lite tydligare programkod så skall vi göra detta i en egen funktion. Vi lämnar därför tillfälligt funktionen *raknaUtMedelvarde* och skapar en ny funktion kallad *medelVarde*.

Funktion medelVarde

Börja med att skapa funktionskroppen:

```
function medelVarde(celler){
//här kommer vi att fylla på med kod.
}
```

Denna funktion tar emot variabeln *celler* som inparameter. Variabeln innehåller tabellceller för den aktuella raden. Det första vi skall göra är att deklarera en variabel kallad *summa* som skall hålla reda på summan av alla cellernas temperaturvärden.

```
var summa = 0;
```

Då cellerna ligger i en array så behöver vi iterera över dem i en loop. Nu måste vi ta hänsyn till hur tabellcellerna ser ut för varje rad. Om vi t.ex. tar den första raden så har den följande utseende:

London	6	18	12	22	23	16.2

Variabeln *celler* innehåller denna rad (array). Raden består av sju celler. Den första är stadens namn (celler[0]) och sedan följer fem celler med temperaturvärden (celler[1-5]) och den sista cellen (celler[6]) skall innehålla medelvärdet av de fem temperaturvärdena (16.2). Nu ska vi göra denna medelvärdesberäkning. Först deklarerar vi en variabel *summa* och därefter drar vi igång beräkningen i form av en iteration över cellerna 1-5. Vi ger därför styrvariabeln *i* startvärdet 1. På så sätt hoppar vi över den första cellen (0) med stadsnamnet. Villkoret för loopen är: i < celler.length -1 vilket betyder att loopen skall iterera så länge som i är mindre än längden på raden (7) – 1, d.v.s. 6. På så sätt avslutar vi loopen innan vi når den sjunde och sista cellen.

```
var summa = 0;
for(var i = 1; i < celler.length -1; i++){

}
```

För varje varv skall vi göra två saker. Först skall vi hämta ut det aktuella temperaturvärdet och placera detta i en variabel kallad *nuvarandeVarde*. Därefter skall vi addera detta värde till summan. Observera att temperaturvärdet ligger i form av en textsträng när vi hämtar ut den med *nodeValue*. Vi måste därför omvandla den från en textsträng till ett heltal med funktion *parseInt()*. Observera att summa tilldelas sitt föregående värde adderat med det nuvarande värdet.

```
var nuvarandeVarde = celler[i].firstChild.nodeValue;
summa = summa + parseInt(nuvarandeVarde);
```

Efter att loopen är klar skall vi beräkna medelvärdet och returnera det.

```
var medel = summa / 5;
return medel;
```

Detta skulle kunna ha gjorts som en programsats: return summa/5; men för tydlighetens skull delar vi upp den i två delar. Den slutliga funktionen blir:

```
function medelVarde(celler){
   var summa = 0;
   for(var i = 1; i < celler.length -1; i++){
      var nuvarandeVarde = celler[i].firstChild.nodeValue;
      summa = summa + parseInt(nuvarandeVarde);
   }
   var medel = summa / 5;
   return medel;
}
```

Tillbaka till raknaUtMedelvarde

Gå nu tillbaka till *raknaUtMedelvarde* och skriv anropet till *medelVarde*. Vi bifogar den aktuella tabellraden (*tabellceller*) till funktionen som parameter. Medelvärdet som returneras lagras i en variabel kallad *medel*.

```
var medel = medelVarde(tabellceller);
```

Du kan i din kod tillfälligt skriva en alert-ruta som visar beräkningen: *alert(medel)*. Med medelvärdet till handa skall vi skapa ett ny textNod som vi skall lägga in den sista cellen på den aktuella raden (tabellceller[6]). Den nya textnoden får som innehåll texten från medelvärdet.

```
var medeltext = document.createTextNode(medel);
```

Nu skall vi lägga till textnoden i cellen. Cellen får vi tag i genom att ange plats 6. Men för att göra funktionen lite mer generell så anger vi att vi vill få ut den sista cellen (vilket är 6). Fördelen med detta är att om man vill förändra antalet kolumner så fungerar fortfarande funktionaliteten. Vi tar ut cellen och låter variabeln medelelement referera till denna.

```
var medelelement = tabellceller[tabellceller.length - 1];
```

Till sist lägger vi in textNoden *medeltext* som ett barnelement till denna cellnod, m.h.a. *appendChild*

```
medelelement.appendChild(medeltext);
```

Funktionen raknaUtMedelvarde är nu klar och har följande utseende:

```
function raknaUtMedelvarde(){

   var tabell = document.getElementById("temperaturtabell");
   var rader = tabell.getElementsByTagName("tr");

   for(var i = 1; i < rader.length; i++){
      var tabellceller = rader[i].getElementsByTagName("td");
      var medel = medelVarde(tabellceller);
      var medeltext = document.createTextNode(medel);
      var medelelement = tabellceller[tabellceller.length - 1];
      medelelement.appendChild(medeltext);
   }
}
```

Nu är första delen klar och ditt JavaScript kan räkna ut och visa medelvärdena i tabellen. Prova att det fungerar.

Ett tips när du skriver JavaScript -kod är att testköra ditt HTML-dokument med en felhanterare som säger till om det är fel i ditt JavaScript. I webbläsaren Firefox klickar du på menyn Verktyg/Tools och väljer därefter menyvalet *Felkonsol*. Information om hur du använder felkonsollen hittar du i kursens filsamling, sektion 2.

Del två – sök värden

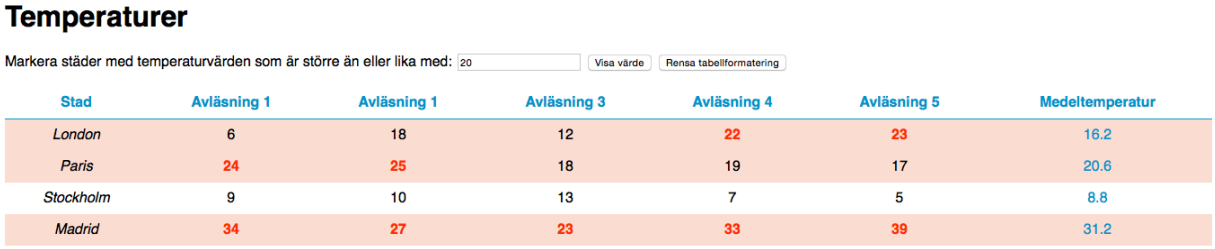
I denna del ska vi implementera en funktion som låter användaren skriva in ett temperaturvärde i inmatningsfältet ovanför tabellen. När användaren klickar på knappen ”Visa värde” skall ditt JavaScript jämföra samtliga temperaturvärden i tabellen med det inmatade värdet. Om ett temperaturvärde hittas som är högre eller lika med det inmatade värdet så skall denna cell markeras med ett klass-attribut *emptd*. Klassen *emptd* finns definierad i CSS-dokumentet *temperatur.css*:

```
td.emptd {
   font-weight: bold;
   color: #ff3300;
}
```

Cellens temperaturvärde kommer att visas med fet, röd stil. Utöver detta skall även den rad (tr-element) som innehåller detta värde markeras med klass-attributet ”emp” (eller rättare sagt alla celler (td-element) i denna rad. CSS-formateringen nedan får till effekt att den aktuella raden får en rosa bakgrund.

```
tr.emp td {
   background: rgb(255, 223, 213);
}
```

Om användaren t.ex. matar in värdet 25 blir resultatet som i bilden nedan:



Figur 3: Formatering av tabellen efter sökning på ett inmatat temperaturvärde.

Två rader innehåller ett eller flera temperaturvärden som är lika med eller större än 25 (Paris och Madrid).

Börja med att skapa en ny funktion kallad hitta(). Denna funktion anropas när man trycker på knappen ”Visa värde”.

```
<input type="button" onclick="hitta();" value="Visa värde"/>
```

Skapa funktionskroppen:

```
function hitta() {

}
```

Det första vi skall göra är att ta fram det inmatade värdet från formuläret. Inmatningsfältet har id ”varde”. Det kan använda genom att ta fram elementet m.h.a. *getElementsById*. Själva värdet får vi fram med *.value*. Eftersom det inmatade är en textsträng måste vi omvandla det till ett heltal m.h.a. *parseInt()*. Slutligen så sparar vi värdet i en variabel kallad *varde*.

(Här skulle det vara lämpligt med en kontroll av att det är siffror som har skrivits in, men vi hoppar över detta och lämnar det som en övning till dig.)

```
var varde=parseInt(document.getElementById("varde").value);
```

Därefter så tar vi fram tabellraderna på samma sätt som vi gjorde tidigare:

```
var tabell=document.getElementById("temperaturtabell");
var rader=tabell.getElementsByTagName("tr");
```

Vi måste nu gå igenom alla rader för att jämföra värden.

```
for(var i=0; i < rader.length; i++){

}
```

Det första vi gör i varje varv är att ta ut den aktuella radens tabellceller.

```
var tabellceller=rader[i].getElementsByTagName("td");
```

Därefter så gör vi en jämförelse. För detta implementerar vi en fristående funktion som jämför det inmatade värdet ”varde” med en hel rad. Vi lämnar funktionen *hitta* tillfälligt och skapar en ny funktion kallad *harVarde()*:

Funktionen harVarde

Skapa funktionskroppen:

```
function harVarde(varde, celler){}
```

Denna funktion tar två inparametrar: *varde* som är det värde vi läste från formuläret och *celler* som är alla celler (td-element) för en aktuell rad. Det första vi gör är att deklarera en boolsk-variabel med namnet *harVarde* (En boolsk variabel kan ha värdet *true* eller *false*). Den har till uppgift att ange om vi har hittat det sökta värdet (eller ett större värde) i den aktuella raden. Vi ger denna startvärdet false.

Därefter skall vi gå igenom alla celler som innehåller temperatur värden. På samma sätt som vi gjorde ovan måste vi hoppa över den första och sista cellen.

```
var harVarde = false;
for(var i = 1; i < celler.length -1; i++){

}
```

Det första vi gör i varje varv är att ta fram temperaturvärdet och omvandlar det till ett heltal:

```
var nuvarandeVarde = parseInt(celler[i].firstChild.nodeValue);
```

Därefter skall vi göra själva jämförelsen. Vi kontrollerar om nuvarande värde (den aktuella cellens temperaturvärde) är större eller lika med *varde* (det inmatade värdet). Om så är fallet sätter vi den logiska flaggan *harVarde* till true. Därefter sätter vi klass-attributet till ”emptd” för denna cell.

```
if(nuvarandeVarde >= varde){
   harVarde = true;
   celler[i].className = "emptd";
}
```

När vi har gått igenom alla celler och loopen är slut avslutar vi funktionen med att returnera sanningsvärdet harVarde. Den slutliga funktionen blir då:

```
function harVarde(varde, celler)
{
   var harVarde = false;
   var sum = 0;

   for(var i = 1; i < celler.length -1; i++)
   {
      var nuvarandeVarde = parseInt(celler[i].firstChild.nodeValue);

      if(nuvarandeVarde >= varde)
      {
         harVarde = true;
         celler[i].className = "emptd";
      }
   }
   return harVarde;
}
```

Tillbaka till hitta

Vi går nu tillbaka till loopen i funktionen *hitta* och skriver anropet till *harVarde*:

```
if(harVarde(varde, tabellceller))
{
   rader[i].className = "emp";
}
```

Anropet skrivs i if-satsen. Detta går bra då *harVarde* returnerar ett sanningsvärde. Villkoret i if-satsen blir då antingen sant eller falskt. Om det blir sant, d.v.s. det fanns ett eller flera temperaturvärden i den aktuella raden som var större eller lika med det inmatade värdet skall vi sätta den aktuella tabellradens klass-attribut till ”emp”. Funktionen är nu färdig:

```
function hitta()
{
   var varde = parseInt(document.getElementById("varde").value);
   var tabell = document.getElementById("temperaturtabell");
   var rader = tabell.getElementsByTagName("tr");


   for(var i = 0; i < rader.length; i++)
   {
      var tabellceller = rader[i].getElementsByTagName("td");
      if(harVarde(varde, tabellceller))
      {
         rader[i].className = "emp";
      }
   }
}
```

Del tre – rensa sökformatering

Ett problem återstår att lösa. När vi har gjort en sökning som ger positivt resultat, d.v.s. vi hittar ett eller flera värden, sitter formateringen kvar. Om vi gör en ny sökning tas den gamla formateringen inte bort. Vi skall därför implementera en funktion som går in och rensar denna formatering.

Skapa en funktion kallad rensa():

```
function rensa()
{
}
```

Den ska gå igenom alla rader och ta bort de klass-formateringar som eventuellt har gjorts. Om ingen sökning har gjorts förut finns ingen formatering. Det finns inte heller någon formatering om vi gjorde en sökning där vi inte hittade något eller några matchande värden.

Vi börjar med att ta fram raderna (*efter att du har gjort denna tutorial, kan en lämplig övning vara att bryta ut dessa rader ur alla funktioner och implementera dessa som en egen funktion*).

```
var tabell = document.getElementById("temperaturtabell");
var rader = tabell.getElementsByTagName("tr");
```

Därefter går vi genom alla rader i en loop:

```
for(var i = 0; i < rader.length; i++)
{
}
```

Det första vi gör i varje var är att vi skriver eventuell klass-formatering med ett tomt värde:

```
rader[i].className = "";
```

Sedan behöver vi gå igenom alla celler och ta bort eventuella cellformateringar. Vi gör detta genom att helt enkelt skriva över alla tabellers klass-attribut med ett tomt värde. Observera att vi nu måste utföra en inre loop för att gå igenom dessa celler. För detta behöver vi använda en annan styrvariabel (j) i den inre loopen. Resultatet blir att för varje varv som den yttre loopen utför så utför den inre loopen fem varv.

```
for(var j = 1; j < celler.length - 1; j++)
{
   celler[j].className = "";
}
```

Vi hade kunnat bryta ut denna inre loop till en egen funktion, som vi har gjort tidigare i lösningen, men det kan vara bra att testa på att använda s.k. nästlade loopar (loopar i loopar). Den slutliga funktionen blir:

```
function rensa()
{
   var tabell = document.getElementById("temperaturtabell");
   var rader = tabell.getElementsByTagName("tr");
   for(var i=0; i < rader.length; i++)
   {
      rader[i].className="";
      var celler = rader[i].getElementsByTagName("td");

      for(var j = 1; j < celler.length - 1; j++)
      {
         celler[j].className = "";
      }
   }
}
```

Vi behöver nu skriva anropet till rensa-funktionen. Den skall anropas i två sammanhang. Vi varje sökning skall en rensning ske innan själva sökningen utförs. Vi lägger därför in anropet till rensa som första rad i *hitta()*, den blir nu:

```
function hitta()
{
   rensa();
   var varde = parseInt(document.getElementById("varde").value);
   var tabell = document.getElementById("temperaturtabell");
   var rader = tabell.getElementsByTagName("tr");

   for(var i = 0; i < rader.length; i++)
   {
      var tabellceller = rader[i].getElementsByTagName("td");
      if(harVarde(varde, tabellceller))
      {
         rader[i].className = "emp";
      }
   }
}
```

Det andra anropet ligger redan förberett i HTML-koden. Om man klickar på knappen ”Rensa tabellformatering” anropas funktionen:

```
<input type="button" onclick="rensa();" value="Rensa tabellformatering" />
```

Kritik och vidare arbete

Det finns några aspekter på ovanstående lösning som bör diskuteras. I kursboken framhåller författaren att en JavaScript-baserad lösning skall ha följande två egenskaper:

1. Lösningen skall vara ***unobtrusive*** (se sidan 65 i kursboken, sida 86 i gamla upplagan), d.v.s. inte påträngande, tillbakadragen, vilket i JavaScript-sammanhang betyder att html-koden skall vara fri ifrån JavaScript-kod. Syftet med det är att skilja innehåll (HTML), formatering (CSS) och funktionalitet (JavaScript) ifrån varandra. Det enda som får finnas är länkarna till JavaScript-filerna.

   Det bryter vi emot i denna tutorial då de två input-fälten har kopplingar till JavaScript genom att vi kopplar händelsen *onclick* till JavaScript-funktioner. Vi kan lösa det med att registrera händelsehanteraren i samband med att funktionen *raknaUtMedelvarde()* körs vid inläsningen av sidan.
2. Lösningen skall tillåta ***graceful degradation*** (se sidan 61 i kursboken, sida 81 i gamla upplagan) som kan översättas ungefär till stilfull försämring, ett konstigt uttryck som i JavaScript-sammanhang betyder att en webbsida ska fungera även om JavaScript är avstängt. Naturligtvis går man miste om fördelen med JavaScript-funktionaliteten, men man skall inte bli berövad all funktionalitet och man skall inte heller drabbas av felaktig data. Det betyder också att om JavaScript är avstängt skall inte HTML-element som är direkt kopplade till JavaScript-funktionaliteten visas.

   Vi bryter mot detta på två sätt i ovanstående lösningen. Tabellkolumnen för medelvärde förblir tom om man slår av JavaScript, vilket kommer att upplevas som förvirrande för användaren. Formuläret ovanför tabellen har också den endast en mening om JavaScript är påslaget. Vi kan lösa detta genom att ta bort formuläret och medelvärdeskolumnen ur HTML-dokumentet och istället skapa den m.h.a. DOM-metoden *createElement* i samband med att funktionen *raknaUtMedelvarde()* utförs (vid inläsningen av sidan). Koden blir då naturligtvis mer komplicerad men lösningen blir bättre. Detta är lämpligt som en egen övning.