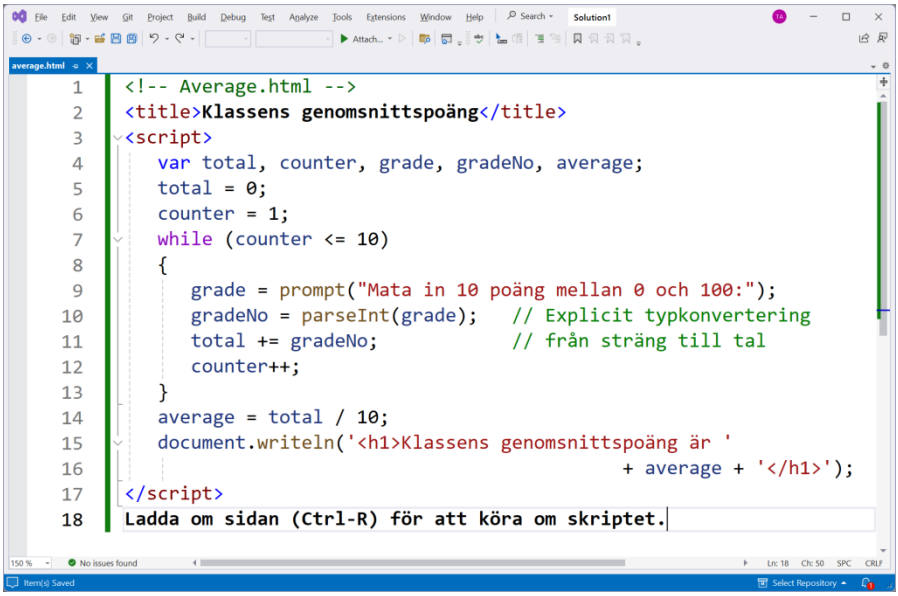
3.7 Räknar-styrd repetition

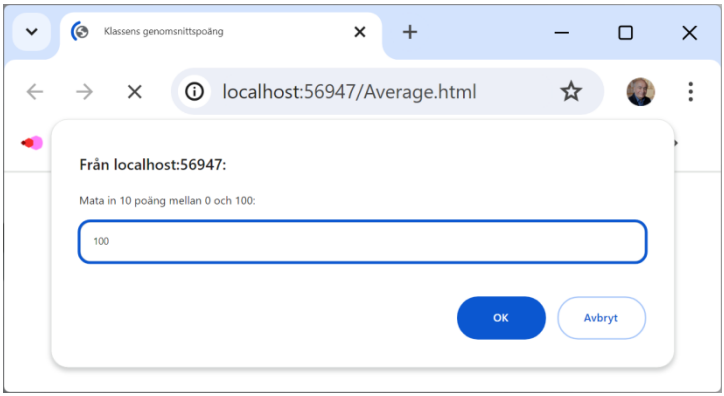
Som exempel för repetitioner väljer vi **while**-satsen. Programmet **Average** kombinerar summeringen av termer (programmet **Sum_while**, sid 60) med inläsning av data. Man beräknar genomsnittet (medelvärdet) av elevernas poäng i en klass:



Samtidigt introduceras dubbeloperatorerna **+=** och **++** i JavaScript som vi känner till från C++. De gör förstås samma sak här som där: **+=** adderar först sina operander och tilldelar sedan resultatet till operanden till vänster. Dubbeloperatorn **++** ökar sin operand med 1. Det som man måste se upp med här, är att de involverade variablerna **total** och **counter** måste initieras innan de uppdateras på raderna **11** och **12**. Initieringen som sker på raderna **5** och **6**, är nödvändig, eftersom deras gamla värden används innan de uppdateras.

Ett körexempel efter 10 inmatningar 100, 88, 93, 55, 68, 77, 83, 95, 73 och 62 ger:

Rutan ovan är den sista efter 10 inmatningsdialoger som genereras av JavaScript-funktionen **`prompt()`** på rad **`9`**:



Vi vet från tidigare att **`prompt()`** returnerar en sträng som vi på rad **`10`** omvandlar till tal med en annan JavaScript-funktion, nämligen **`parseInt()`**. Tekniken kallas för *explicit typkonvertering*, dvs självgjord omvandling av datatypen, till skillnad från *automatisk typkonvertering* som JavaScript utför p.g.a. vissa regler som är inbyggda i språket. Här är den explicita varianten nödvändig.

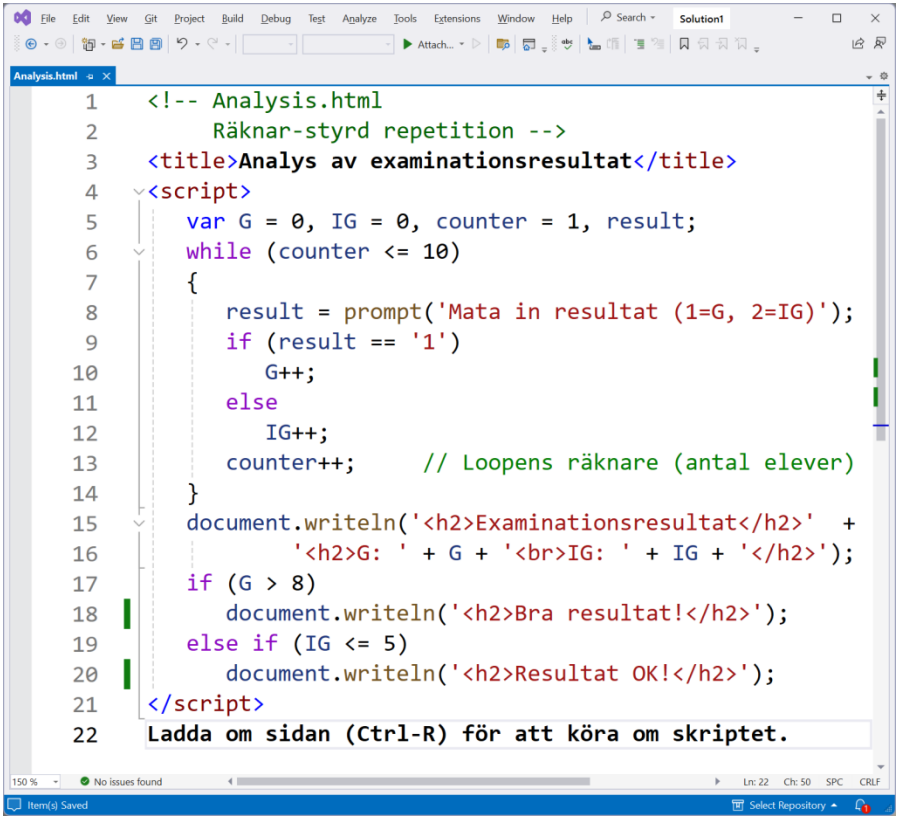
En annan nyhet i programmet **`Average`** på förra sidan är att vi för första gången deklarerar våra variabler med **`var`** på rad **`4`**, vilket inte är obligatoriskt i JavaScript. Vi kommer att fortsätta med det, när våra program blir längre och behöver struktureras mer. Strukturering är redan avgörande i detta programexempel, speciellt vad gäller frågan, vilka delar av koden måste stå *före*, vilka *i* och vilka *efter* loopen:

Observera att både inläsningen och summeringen av data, inkl. typkonverteringen görs *i* **`while`**-loopen, mellan allt annat står utanför den, speciellt beräkningen av genomsnittspoängen (medelvärdet) som står *efter* loopen på rad **`14`**.

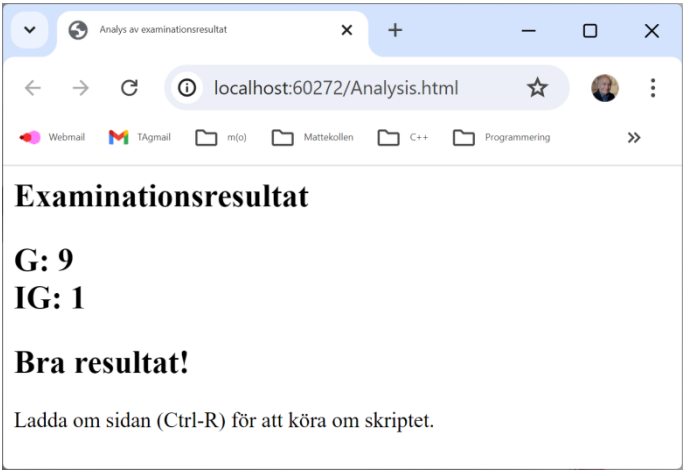
En speciell roll spelar variabeln **`counter`** som tar reda på antal varv. Den initieras *före* och uppdateras *i* loopen, för att användas i avslutningsvillkoret och förhindra evighetsloop. Det är **`counter`**n och antalet 10 som vi har fastskrivit i koden som styr **`while`**-loopen och därmed antalet inmatningar. Man talar om räknar-styrd repetition.

Analys av examinationsresultat

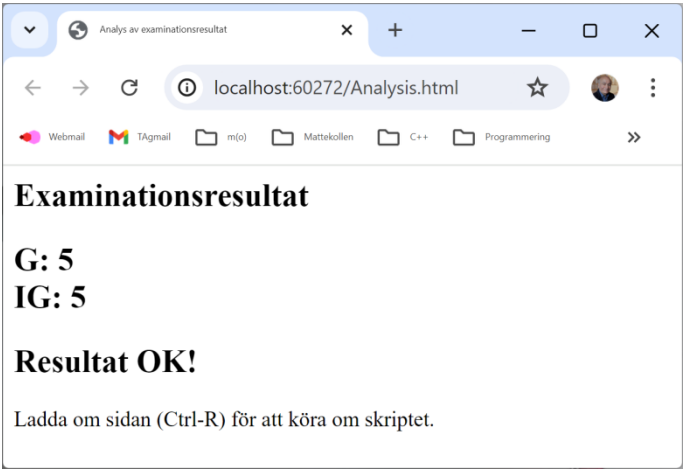
Ett annat exempel på räknar-styrd repetition är programexemplet **`Analysis`** på nästa sida. Här försöker man att få en sammanfattning eller analys av 10 elevers resultat i ett prov. Beroende på analysen som baseras på en policy skrivs ut vissa slutsatser.



En körning med 10 inmatningar 1, 2, 1, 1, 1, 1, 1, 1, 1 och 1 ger:



Medan en annan körning med inmatningarna 1, 2, 1, 2, 2, 1, 2, 2, 1 och 1 resulterar i:

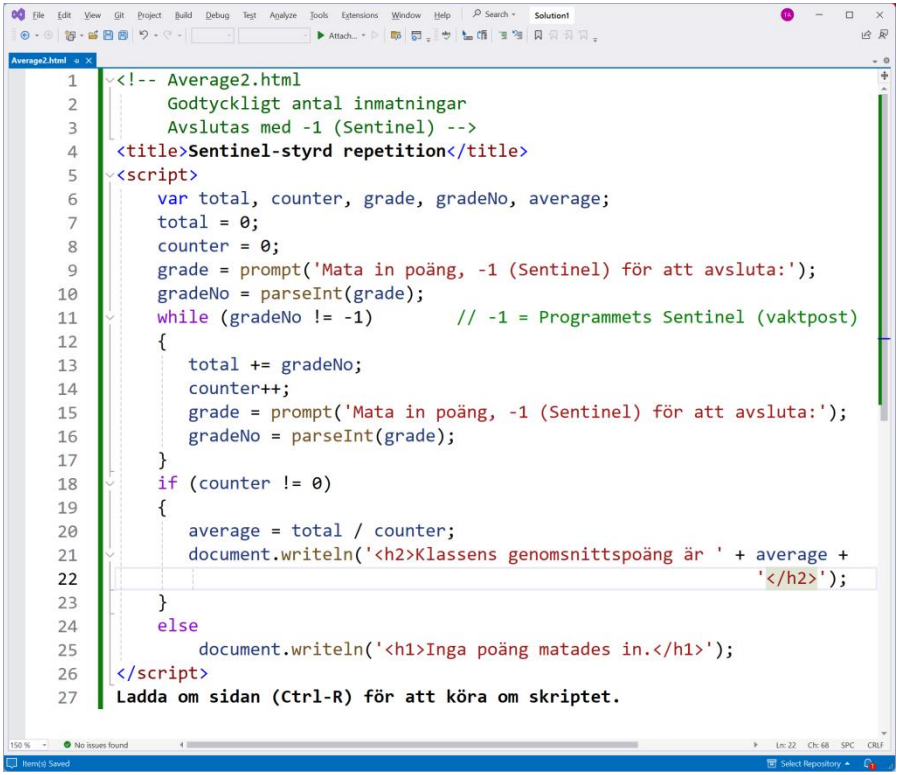


En övning i räknar-styrd repetition vore att byta ut loopens räknare **`counter`** i programmet **`Analysis`** med summan **`G + IG`** av alla poäng, för att spara en variabel och förkorta koden. Ev. blir det nödvändigt att även justera **`while`**-loopens avslutningsvillkor. Denna uppgift är överlåten till övn 3.15 (sid 76).

3.8 Sentinel-styrd repetition

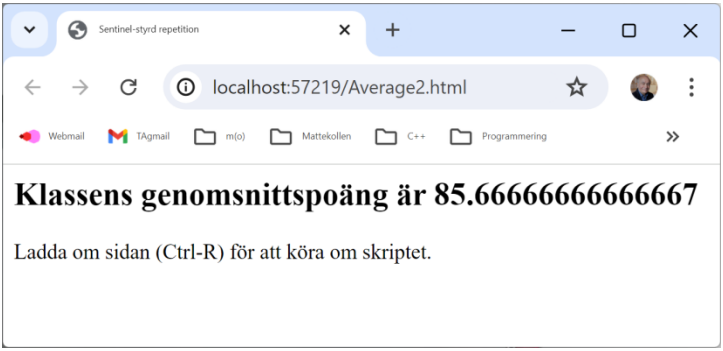
Programmet **Average** (sid 62) har en stor nackdel: Antalet inmatningar är statiskt. **while**-loopens antal varv är hårdkodat. Bara klasser med 10 elever kan använda programmet. Självklart är det önskvärt att användaren ska kunna bestämma antalet inmatningar, inte vi som kodar. Programmet borde vara användbart för alla möjliga storlekar på klasser.

Nu vill vi generalisera programmet **Average** genom att låta antalet inmatningar vara fritt väljbart. Programmet **Average2** som följer, är ett exempel på en lösning av detta problem. Det som skiljer dessa två program är i huvudsak loopens avslutningsvillkor, dvs sättet att styra repetitionen. **Average**:s **while**-loop styrs av en räknare – variabeln **counter** – som successivt uppdateras. Repetitionen är räknarstyrd. **Average2**:s repetition däremot styrs av ett speciellt *värde* som kallas för *Sentinel* (vaktpost) som är nyckeln till att avsluta programmet. Man pratar om *Sentinel-styrd repetition*. Detta värde som bestäms i koden, får inte finnas bland programmets ”legitima” data. Vad vi menar med det kommer vi snart att se.



Som man ser har vi valt värdet *-1* som *Sentinel*. *-1* kan inte vara en elevpoäng.

Matar vi t.ex. in: 97, 88, 72 och slutligen *-1* får vi följande utskrift:



Inmatningen av *-1* avslutar körningen därför att programmets Sentinel har valts till *-1* på rad **11**. Decimaltalet är resultatet av divisionen på rad **20**.

Matar vi in däremot *-1* från början blir det följande utskrift:

Anledningen är att vi p.g.a. **`gradeNo`** = **`-1`** aldrig kommer in i **`while`**-loopen och p.g.a. **`counter`** = **`0`** hamnar i **`else`**-delen av **`if-else`**-satsen på rad **25**.

Logiken bygger på programmets struktur: en **`while`**-loop följd av en **`if-else`**-sats. Dessutom nödvändigheten att ha inläsningen av data *två gånger* i programmet, en gång *före* (rad **9**) och en gång *i* **`while`**-loopen (rad **15**). Före loopen, för att kunna bilda loopens avslutningsvillkor. I loopen, för att kunna mata in data flera gånger. Och faktiskt gäller även det omvända: Programmets struktur är vald för att just implementera den önskade logiken.

Nödvändigheten av *två* inläsningar kan anses som en nackdel av programmet. Frågan är: kan man hitta en alternativ struktur som möjliggör endast *ett* inläsningstillfälle, t.ex. genom att byta till en annan loop-typ? Frågan är föremål för en övning, se övn 3.16 (sid 76).

3.9 HTML-element i loopar

För att förstå hur HTML-element fungerar i loopar vill vi börja med att titta på HTML-element utan loopar och sedan fortsätta att sätta dem i loopar. Vi tar som exempel **p**-elementet där **p** står för paragraf dvs stycke. Men vad är ett *element* i HTML? Det blir en liten repetition i HTMLs grunder från *Webbutveckling 1*:

HTMLs taggar, innehåll, element och attribut

En *tagg* i HTML inleds med **<** och avslutas med **>** . Taggar används i regel parvis: en *starttagg* markerar *början* och en *starttagg* markerar *slutet*. Ex.:

```
<p>Välkommen till HTML!</p>
```

Raden ovan har två taggar: starttaggen ** ** och sluttaggen ** **.

Det som står mellan start- och sluttagg, texten **Välkommen till HTML!**, kallas för *innehåll*, närmare bestämt innehåll till **p**-elementet.

p-elementet skapar ett nytt stycke (paragraf) i textflödet som inkluderar radbyte efteråt. Innehållet skrivs ut i paragrafen.

Generellt har ett *element* i HTML följande ingredienser:

Starttagg + innehåll + sluttagg = **Element**

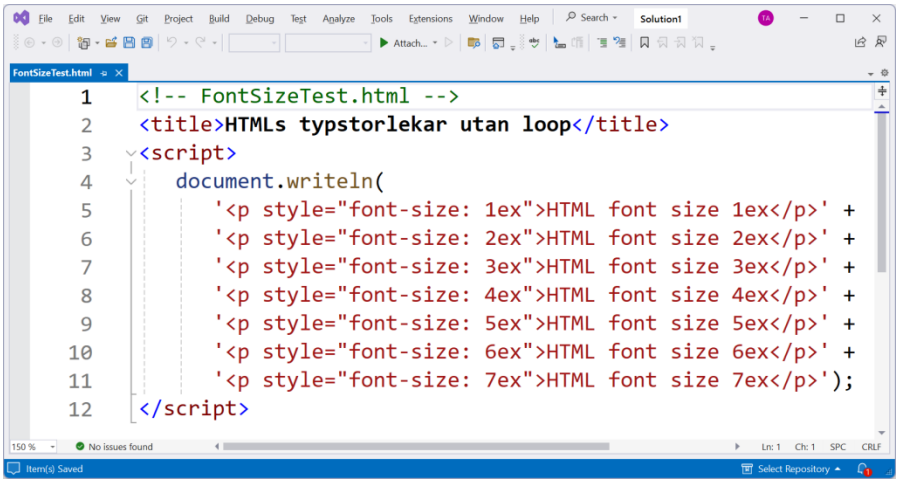
Ett annat exempel på ett **p**-element som är lite mer invecklat är:

```
<p style="font-size: 4ex">HTML font size 4ex</p>
```

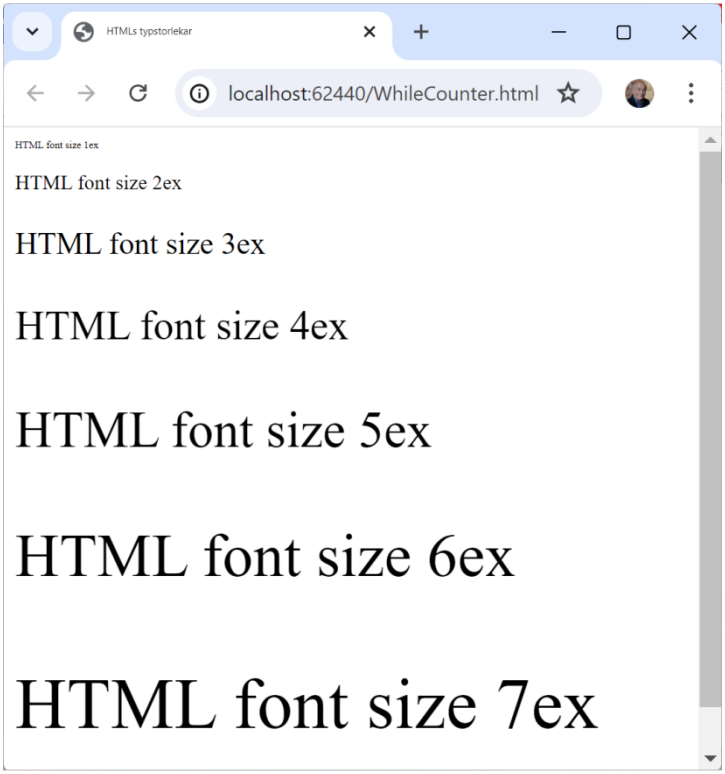
Innehållet i det här **p**-elementet som skrivs ut, är texten **HTML font size 4ex**, eftersom det är den som står mellan start- och sluttagg. Men starttaggen har fått ett nytt utseende: den har utvidgats och fått ett s.k. *attribut*. Attributets *namn* är **style** och attributets *värde* är **"font-size: 4ex"**. Värdet måste anges som sträng dvs inom citationstecken. I exemplet bestämmer **style**-attributets värde att texten som skrivs ut, ska ha HTML-typsnittet **4ex**. Både attributets namn och värde är del av **p**-elementet, koden som står *inom* taggarna, medan innehållet är texten som ska skrivas ut och står *utanför* taggarna.

Vilka typsnitt som finns i HTML och hur de ser ut, ska vi snart titta på.

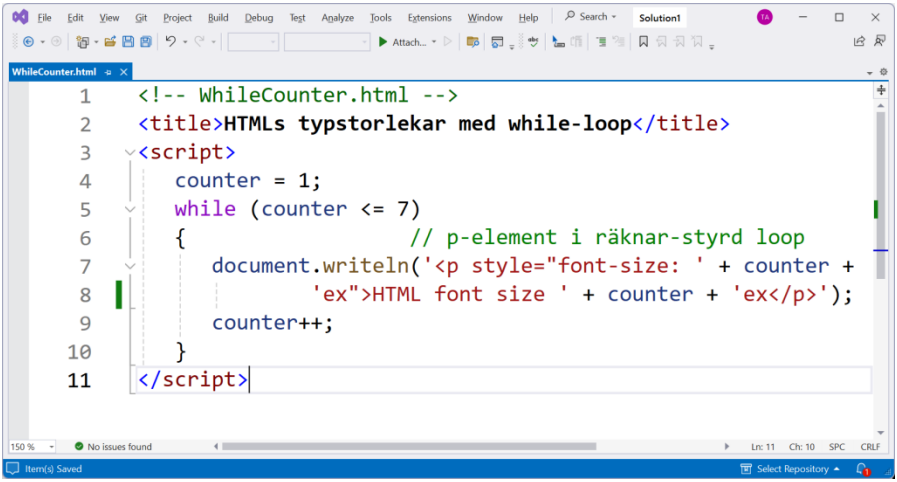
I scriptet **FontSizeTest** på nästa sidan skriver vi ut alla 7 typsnitt utan loop. Sedan ska vi gå över att skriva om koden till en **while**-loop.



Här har vi tagit över **p**-elementets syntax som förklarades på förra sidan. Koden skriker efter att skrivas om till en loop, vilket vi tar upp på nästa sida. En körning av scriptet **FontSizeTest** visar HTMLs 7 typstorlekar:



I scriptet **WhileCounter** har vi effektiviserat koden med en **while**-loop som sträcker sig över raderna **5-10**. Loopen styrs av räknaren **counter**.



Den svåraste delen av koden utgörs av raderna **7-8**, i **document.writeln**-parentesen. Problemet består av att vi måste baka in typsnittens storlekar 1-7 som representeras av räknaren **counter**, i den text som **p**-elementets innehåll ska skriva ut. För att göra det måste vi konkatenera varaiabeln **counter** med vanliga strängar.

Om vi extraherar **document.writeln**-funktionens utskriftssträng som innehåller ett **p**-elementet, och skriver den på en rad, för att analysera den, ser det ut så här:

```
'<p style="font-size: ' + counter + 'ex">HTML font size ' + counter + 'ex</p>'
```

Det som är framhävt med grå bakgrund, är **p**-elementets innehåll som kommer att skrivas ut. Attributet **style**s värde är skrivet inom citationstecken, medan andra strängar har markerats med apostrofer. Detta för att åstadkomma korrekt parning. Några strängar är nästlade i andra. Dessutom överlappar vissa strängmarkeringar med apostrofer andra strängmarkeringar med citationstecken. Överlappning förekommer även i början och slutet med **p**-elementets start- och sluttagg. Alla **+** tecken betyder konkatenering mellan variabeln **counter** och strängkonstanter.

Vid den här konstruktionen har vi dragit nytta av att vi i JavaScript har möjligheten att markera strängar både med citationstecken och apostrofer. Hade det funnits endast ett tecken för strängmarkering, hade vi varit tvungna att använda *escapesekvensen* **\"** som alternativ. Testa gärna att ersätta alla apostrofer med citationstecken och de två citationstecknen kring attributets värde med **\"**. Vi har gjort så i nästa script. Escapesekvenser har i JavaScript samma *betydelse* som i C++. Om de *fungerar* i alla sammanhang beror ofta på samspelet mellan JavaScript och HTML.

Körresultatet av scriptet **WhileCounter** är identisk med **FontSizeTest**s resultat på förra sidan.