Ulf Tornert
ulftt@gy.varmdo.se

En introduktion till JavaScript

Ulf Tornert
ulftt@gy.varmdo.se

JavaScript – en introduktion

Vad är JavaScript?

Till att börja med så ska vi göra oss av med en vanlig missuppfattning: JavaScript är *inte* Java och JavaScript har i alla avseenden ganska lite med Java att göra. JavaScript är, som namnet antyder, ett skriptspråk och dess tillämpningar ”lever och frodas” endast *inuti* en webbläsare. Det går alltså inte att exempelvis skapa ett självständigt, exekverbart program av ett JavaScript, utan man *bäddar in* alla skript i en webbsida och dessa exekveras sedan av webbläsaren. Ett förbehåll är ju förstås att den aktuella webbläsaren har en JavaScript-tolk, och det har ju inte alla! I skrivande stund kan man väl säga att webben domineras av Netscape Navigator och Internet Explorer. Båda dessa hanterar JavaScript – om än lite olika som vi senare bittert kommer att erfara! Dessutom påstås det att Opera numera hanterar JavaScript precis lika bra som de två föregående.

Man säger att JavaScript är ett *klientbaserat* skriptspråk. Med detta menas att alla skript exekveras på klientens (läs: besökarens) dator. Ett skript manipulerar i något avseende de objekt som finns i en webbsida för att göra den mer *dynamisk* än vad som är möjligt med standard HTML. Några vanliga exempel är: rollover-bilder, formulärkontroller och smartare navigeringsfunktioner med uppdatering av multipla ramar etc. Mer om allt detta längre fram. Det fina med att ha ett klientbaserat skriptspråk är ju att det som händer exekveras på besökarens dator, utan att servern som levererade sidan alls är inblandad. Ni förstår själva vad det kan betyda för trafiken till och från välbesökta sajter…

Det går att skapa nästan vad man vill med JavaScript, eftersom språket förfogar över allt sådant som in- och utmatning av tecken, standardfunktioner för matematiska beräkningar, kontrollstrukturer som t. ex.. *if*…*else*-satsen, egendefinierade funktioner, komparativa operatorer, booleska operatorer etc. etc. Alltså: allt det som vi annars är vana att ett språk, som exempelvis Pascal, har. Det man kan sakna är tillgång till att exempelvis spara information på serverns hårddiskar. Det klarar inte JavaScript, utan där måste man använda andra tekniker. JavaScript är i princip helt ofarligt och ryktena om att s.k. cookies används till att ställa till det för besökaren vill jag avfärda som grundlösa. (Cookie-mekanismen ligger tyvärr utanför den här kursens omfattning – läs mer på egen hand.)

Syntaxen i språket är väldigt lik syntaxen i C++ och Java. Kanske kom namnet JavaScript till för att ”rida lite på vågen”, då Java kom ungefär samtidigt. Skaparen heter Brendan Eich och arbetade vid den aktuella tidpunkten för Netscape Communications Corporation, och denne insåg säkert vikten av att presumtiva webbutvecklare skulle uppleva syntaxen som familjär. Kom dock ihåg att t. ex. Java är ett fullskaligt objektorienterat programmeringsspråk, med vilket du skulle kunna skapa en helt ny webbläsare (inom vilken ett JavaScript skulle kunna exekveras)! Det är därför mycket stor skillnad på de båda.

Vad behöver du för att lära dig JavaScript?

Det är väldigt bra om man redan har goda kunskaper i HTML, eftersom JavaScript i mångt och mycket skapades för att manipulera med ”HTML-objekten” i en webbsida. Däremot behöver man inte kunna någon programmering för att börja. Det finns många, mycket bra böcker för vem som helst att börja med – en sådan är **JavaScript Bible** av **Benny Goodman** på IDG´s förlag. Dessutom finns det ett närmast oändligt antal tutorials på webben att tillgå. Om man redan har kunskaper i C++, Java, eller något annat programmeringsspråk, kommer JavaScript inte att bereda några direkta problem! Sedan behöver du förstås en webbläsare och en texteditor.

Ulf Tornert
ulftt@gy.varmdo.se

Hur JavaScript integreras i en webbsida

Att bädda in skripten i webbsidan

Ett skript bäddas in i en webbsida mellan `<SCRIPT></SCRIPT>` taggarna på följande vis:

```
<HTML>
<HEAD>
<TITLE>En webbsida!</TITLE>

<SCRIPT LANGUAGE=”JavaScript”><!--

// Här kommer JavaScript koden…
// Kod som ligger innanför HEAD-taggarna måste
// anropas från dokumentet – t. ex. från ett OnMouseOver-event
// eller annat funktionsanrop

// --></SCRIPT>
</HEAD>
<BODY>

Här kommer innehållet i webbsidan…

<SCRIPT LANGUAGE=”JavaScript”><!--

// Detta skript körs på den här platsen
// Objekt som skriptet refererar till måste
// vara skapade – annars får du skriptfel!

// --></SCRIPT>

…och här tar sidan slut

</BODY>
</HTML>
```

I ärlighetens namn så innehåller koden ovan mer än vad som faktiskt är nödvändigt. Du är säkert förtrogen med att taggparet `<!-- och -->` definierar en kommentar i HTML. Två stycken slashar `//` definierar en kommentar i JavaScript. Om webbläsaren är för gammal och därmed inte förstår `<SCRIPT>`-taggen kommer den att hoppa över den raden och börja med nästa. Där hittar den en kommentar och hoppar således över hela skriptet, eftersom detta i sin helhet är inbäddat i en HTML-kommentar! Att jag sedan mitt i skriptet infogar en JavaScript kommentar påverkar ju inte det faktum att webbläsaren ser det hela som en enda stor kommentar. På detta vis kan jag någorlunda säkerställa att gamla webbläsare inte genererar fel i min sida, eller att de rent av inte genererar sidan alls…
Jag är dock tvingad att kommentera ut ’`-->`’ annars får JavaScript-tolken i Netscape problem att förstå koden. Det finns också möjlighet att ladda skript från en extern fil. Det kan vara mycket användbart om många sidor på din sajt använder samma skript – då slipper du bädda in dem i alla sidor. Allt du behöver göra är att spara ditt skriptbibliotek med filtillägget ’.js’. Filen får inte innehålla något annat än JavaScript kod – ingen HTML heller, och den skall vara sparad som en textfil. Anta att du har ett skriptbibliotek i namnet ’`mina_skript.js`’, dessa blir tillgängliga för en sida genom att bädda in dem med

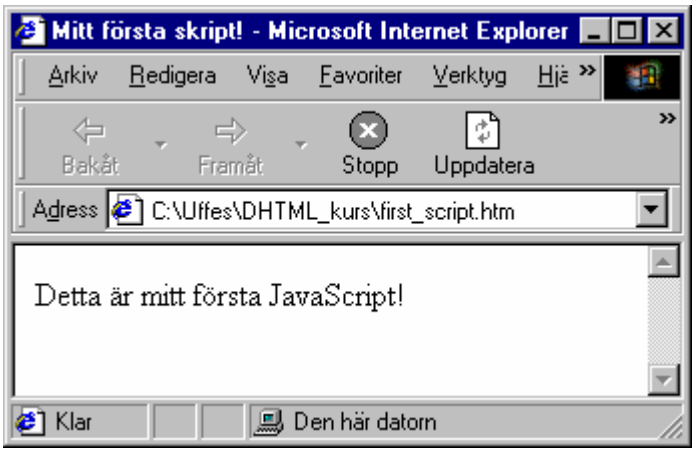
```
<SCRIPT LANGUAGE=”JavaScript” SRC=”mina_skript.js”></SCRIPT>.
```

Ulf Tornert
ulftt@gy.varmdo.se

Vårt första JavaScript
Skriv in följande kod i din editor och läs in filen i webbläsaren!

```
<HTML>
<HEAD>
   <TITLE>Mitt första skript!</TITLE>
</HEAD>
<BODY>

<SCRIPT LANGUAGE=”JavaScript”><!--

document.write(”Detta är mitt första JavaScript!”);

// --></SCRIPT>

</BODY>
</HTML>
```

Detta bör generera en sida som ser ut ungefär så här:



Satsen `document.write(”Detta är mitt första JavaScript!”);` är mycket illustrativ för hur man använder fördefinierade funktioner i JavaScript. Här använder vi funktionen `write()` som tillhör objektet `document`. Funktionen tar en sträng som argument, vilken skall anges inom citationstecken i parentes, och hela funktionsanropet avslutas med ett semikolon. I detta sammanhang är det nog på sin plats att tala något om ”*the Document Object Model*” (DOM).

Kort om syntax och Document Object Model

Document Object Model (DOM)

I JavaScript manipulerar man mycket med de objekt som ingår i dokumentet (=sidan). Det finns ett motsvarande objekt i webbläsaren som heter just `document`. När en webbläsare (åtminstone NN och IE från ver. 4 och uppåt) läser in en sida och återskapar den, så skapar den samtidigt en hel rad variabler till `document`, vilka speglar de olika objekten som ingår i sidan. Om det exempelvis finns tre bilder i sidan kommer det att finnas en vektor – `images[]` – med tre element som innehåller information om dessa bilder. Den information som lagras är exempelvis namnet på bilden, placering och pixelyta etc. Om det finns formulär i sidan kommer vektorn `forms[]` att innehålla information om dessa. Ett formulär i sin tur består ju ofta av många olika delar som exempelvis textrutor, knappar av olika sorter och textfält etc. Information om dessa

lagras också. Om jag har en ’checkbox’ i ett formulär i en sida, och med ett skript vill kontrollera om den är förbockad så måste jag ju ha något sätt att referera till den. Det är detta som DOM är till för. Det finns förstås en viss hierarki mellan objekten som återspeglas i hur man refererar till dem. Generellt kan sägas att ordningen för hierarkin är:

1. Webbläsarfönstret självt (`window-object`)
2. Eventuellt en eller flera ramar (`frame-object`)
3. Dokumentet (`document-object`), tillsammans med `location` och `history`
4. Objekt i sidan t. ex:
   - ✍ länkar (`links[]`)
   - ✍ ankare (`anchors[]`)
   - ✍ lager (`layers[]`, gäller bara NN)
   - ✍ formulär (`forms[]`)
   - ✍ Java applets (`applets[]`)
   - ✍ bilder (`images[]`)
5. Formulären i sin tur innehåller eventuellt:
   - ✍ textrutor
   - ✍ knappar
   - ✍ select-boxar
   - ✍ etc.

Om jag har namngett ett formulär med `userForm` och en checkbox med `checkID`, kan jag referera till denna genom `document.userForm.checkID`. Här ser man hierarkin klart och tydligt, dessutom ser du att man tar sig längre och längre ner i hierarkin med punkter.

Referenser till objekt i sidan

I exemplet ovan visade jag hur man kan referera till ett namngivet objekt i en sida. Det finns även en mer generell standard för referenser, men dessa kräver att man verkligen vet vad man sysslar med. Dessutom är det så att man kan få skriptfel om man vid ett senare tillfälle ändrar sidans layout! Noggrann eftertanke krävs alltså! Å andra sidan kan man göra helt andra saker med detta andra sätt att referera till objekt – därför måste vi lära oss också detta.

Referenser till namngivna objekt

En bild i en sida vilken jag skrivit in så här: `<img src=”sökväg/bild1.gif” name=”nuna”>` kan jag referera till genom: `document.images[”nuna”]`. Det är sedan fullt möjligt för mig att via ett skript ändra den bildens källfil genom följande sats:

```
document.images[”nuna”].src = ”bild2.gif”;
```

Det är även tillåtet att referera till bildens källfil genom `document.nuna.src`. Tänk på att bilden måste ha ett unikt namn! På liknande sätt refererar jag till textrutan `userName` i formuläret `userInfo` med `document.userInfo.userName`. Sedan kan jag hämta texten där genom

```
name = document.userInfo.userName.value;
```

I nyare versioner av webbläsare är det vanligare att använda standardmetoden `getElementById()` istället. Anta att du vill komma åt bilden `nuna` (ovan) för att ändra källfilen till den. Du kan då skriva följande

```
document.getElementById("nuna").src = "bild2.gif";
```

Ulf Tornert
ulftt@gy.varmdo.se

Detta är ett ganska avancerat uttryck så vi tar upp det senare igen. Du kan nu själv välja vilket sätt du väljer att arbeta med. Fördelen med metoden `getElementById()` är att den skall vara gemensam för alla nyare webbläsare enligt nån slags standard som webbkonsortiet (W3C) står för.

Referenser till objekt generellt

Om jag vill ändra källfil till en bild och jag vet att det är den *tredje* bilden i dokumentet, så kan jag åstadkomma detta genom:

```
document.images[2].src = ”bild2.gif”.
```

Lägg märke till att det står siffran 2 istället för 3 inom hakparenteserna. Det är så därför att JavaScript, liksom C++ och Java, har index 0 som första index i en vektor. Den första bilden har således referensen `document.images[0]`. Samma förfarande gäller länkar, formulär, ramar och så vidare. Om dina sidor är uppbyggda med ramar och du vill ladda in en ny sida i ram nr 2 kan du skriva på två sätt:

```
window.frames[1].location = ”webbsida.htm”.
window.frames[1].location.href = ”webbsida.htm”.
```

En referens till en textruta i ett ramverk kanske skulle kunna se ut så här:

```
name = window.frames[1].document.forms[0].userName.value;
```

Dvs `name` får värdet av innehållet i textrutan `userName` i det första formuläret i den andra ramen i ett ramverk.

Referens till språket JavaScript

Denna referens kommer endast att behandla det mest grundläggande om syntax och objekt i JavaScript. Dessutom skall sägas att de flesta objekt och metoder vi går igenom är kompatibla med NN4+ och IE4+. Det vill säga att några av dem kanske är kompatibla med versionsnummer 3 (eller t.o.m. mindre), men ta det inte för givet. Om du vill bli duktig på att göra skriptade sidor *måste* du skaffa en heltäckande referens i ämnet. I sådana står det också klart och tydligt vilka objekt och metoder som hör till vilken webbläsargeneration etc. Jag föredrar själv att använda en bok framför att surfa fram samma information på webben – det kanske är en vanesak och möjligtvis börjar jag bli gammalmodig… En riktigt bra bok i ämnet kanske kostar runt 400 – 500 kr, men det är det definitivt värt!

Variabler

Variabler i JavaScript är ’löst typade’ – vilket betyder att de inte har någon entydigt bestämd datatyp. Ni som har programmerat förut vet att alla variabler måste *deklareras* med en viss typ, för att sedan (eller samtidigt) tilldelas ett lämpligt värde. I exempelvis C++ finns det många olika fördefinierade typer av variabler – några är: heltal (int), decimaltal (float), enstaka tecken (char) och logiska (boolean) variabler. Om jag vill ha en variabel som skall representera ett heltal så kan jag i C++ skriva:

```
int x = 4;
```

Det är sedan möjligt för mig att ändra detta värde genom exempelvis

```
x = x + 1;
```
(x blir alltså lika med heltalet 5). Men att skriva

```
x = ”Detta är en textsträng”;
```

skulle generera ett fel i nästan vilket programmeringsspråk som helst, därför att jag försöker tilldela en heltalsvariabel en *textsträng* (en textsträng är flera tecken efter varandra som i exemplet ovan). Inte så i JavaScript – där är ovanstående helt okej! Alla variabler är ’löst typade’. Senare i denna bok kommer jag att visa några av de stränghanteringsfunktioner som finns i JavaScript. Att hantera strängar är en mycket vanlig programmeringsuppgift, därför har man skaffat en hel uppsättning med standardfunktioner färdiga att använda på vilka strängar som helst. Bekvämt! För att definiera en variabel med namnet `userName` skriver du

```
var userName;
```

För att tilldela denna något värde skriver du exempelvis

```
userName = ”Kalle Katt”; // en sträng
```

Alternativt kan du deklarera- och initiera variabeln samtidigt genom

```
var userName = ”Kalle Katt”;
```

Nyckelordet `var` skriver du endast en gång för varje variabel (då den deklareras). Observera att det nu i JavaScript är tillåtet att skriva `userName = 32.5;`

Du får inte ge dina variabler vilka namn som helst. Godkända namn är de som innehåller s.k. alfanumeriska tecken såsom ’a – ö’, ’A – Ö’, ’1 – 9’ och ’_’. Namnet får inte börja på en siffra dock. Så är t. ex. `userName, user2, fred_flinta` godkända namn på variabler. Icke godkända är `user name` (med mellanslag), `2user, fred@flinta.se` osv. Ta dock för vana att ge variabler namn med tecken från det engelska alfabetet. Anledningen är att risken för skriptfel då minimeras.

På grund av att variabler egentligen har olika typ beroende på vad du tilldelar dem, kommer också *typomvandlingar* att ske automatiskt. Om du till t. ex. adderar heltalet 3 med strängen ”3” kommer du att få ”33” som resultat (en sträng, inte heltal). Här lämnar jag över till en uttömmande referens, eller till din experimentlusta! Om du gärna vill ha ett heltal eller decimaltal kan du använda de fördefinierade funktionerna `parseInt()` och `parseFloat()`, vilka ger heltal respektive decimaltal som resultat. Ett exempel:

`parseInt(”10 Downing Street”)` ger resultatet 10 (heltal).

Vektorer

Vi har redan sett hur man kan referera till den tredje bilden i en sida genom `document.images[2].` Vad betyder egentligen `images[2]`? Jag sa under rubriken ’Referenser till objekt i sidan’ att webbläsaren skapar en *avbildning* av webbsidan där alla objekt finns representerade i *vektorer.* Så kommer information om alla bilder i sidan att finnas representerade i vektorn `images[]`. Vi har alltså att göra med en vektor – men vad är det då? Tänk dig en tabell med ett antal kolumner. I varje kolumn finns ett heltal, och kolumnerna är numrerade med ett *index* från 0 och uppåt i steg om ett. Om radrubriken är `talnr` skulle det se ut så här:

index	0	1	2	3	4	etc.
talnr	**5**	**12**	**25**	**7**	**34**	etc.

Vi skulle kunna referera till talet 25 som ”*talet på raden **talnr** och kolumn **2***”. På programmeringsspråk gör man samma referens genom `talnr[2]`, där `talnr` är en slags tabellvariabel – en vektor – ungefär som radrubriken ovan. Hakparenteserna används alltså för att referera till det aktuella talet i vektorn `talnr`. Så man kan säga att `talnr` är en vektorvariabel som innehåller flera tal och `talnr[2]` är endast ett tal (det tredje) i den vektorn. I *DOM* är `images[]` en vektor med bilder, och `images[2]` är en bild.

När används vektorer och hur skapar man dem?

Vektorer är alldeles ypperliga att använda om skripten som manipulerar med objekten i dem gör det via upprepade slingor. Tänk dig att jag har en vektor `myString` med 10 textsträngar i, och så vill jag skriva ut allt som finns i den vektorn. Istället för att skriva `document.write(myString[index]);` tio gånger – en för varje index så kan jag automatisera hela förloppet med en *for*-slinga:

```
for (var i = 0; i < 10; i++)
{
   document.write(myString[i]);
}
```

Första gången slingan körs har i värdet 0 och därför kommer `myString[0]` att skrivas ut. Nästa gång slingan körs har i:s värde ökats till 1 och därför skrivs `myString[1]` ut osv osv. Hur *for*-slingan fungerar mer ingående läser du under rubriken ’for-slingan’.

Att skapa en vektor

Om jag vill skapa en vektor med namnet `myString` innehållande tio textsträngar skriver jag

```
var myString = new Array(10);
```

(Kom dock ihåg: Eftersom JavaScript är löst typat så är det inget som säger att vektorn måste innehålla just strängar – det står mig fritt att välja innehåll i vektorn precis hur som helst. Detta kan kännas mycket märkligt för alla er som har programmerat förut). För att fylla vektorn med strängar kan jag skriva:

```
myString[0] = ”Kalle Katt”;
myString[1] = ”Dagge Mask”;
myString[2] = ”Gusten Gorilla”;
myString[3] = ”Fia Flodhäst”;
myString[4] = ”Peter Polis”;
etc.
```

När jag har fyllt hela vektorn kan jag alltså exekvera *for*-slingan i föregående avsnitt för att skriva ut allting på en gång. Nu finns det en del att säga om operatorn `new`, med vilken man egentligen reserverar minnesutrymme för att kunna lagra vektorns all information. Detta minnesutrymme kan senare frigöras med operatorn `delete` för att inte få vad man brukar kalla *minnesläckage* – dvs att skripten tar i anspråk minnesutrymme som de senare inte återlämnar. Men i ärlighetens namn kan ni nog fullständigt bortse från sådana eventuella effekter (särskilt som de datorer vi använder har några kilobytes att avvara för våra små enkla skript)…

Ulf Tornert
ulftt@gy.varmdo.se

Aritmetiska operatorer

De fem (?) räknesätten

JavaScript har självklart alla möjligheter till matematiska beräkningar i dina skript. De fyra räknesätten har de gamla välkända operatorerna '+', '-', '*' och '/'. Resultatet från en beräkning beror på datatyperna av de ingående värdena. Jag nämnde det kort under rubriken 'Variabler': Om jag adderar heltalet 3 med strängen "3" får jag resultatet "33" (sträng, inte heltal). Därför kan det vara av intresse att titta på alla fyra operatorerna med dess eventuella resultattyp.

Symbol	Uttryck	a och b kan vara	Resultat
+	a + b	heltal, dec.tal och sträng	heltal, dec.tal och sträng
-	a – b	heltal, dec.tal	heltal, dec.tal
*	a * b	heltal, dec.tal	heltal, dec.tal
/	a / b	heltal, dec.tal	heltal, dec.tal

Det finns en femte mycket vanlig operator (räknesätt) som kallas *modulo*-operatorn. Denna skrivs med '%' – tecknet. Den delar ett tal med ett annat och returnerar *divisionsresten*. Ta exempelvis a = 5 och b = 2. Om du delar a med b får du 2,5 eller egentligen 2 hela och divisionsresten 1. Resultatet av a % b (läses "*a modulo b*") blir därför lika med 1 (resten när du dividerar 5 med 2).

Symbol	Uttryck	a och b kan vara	Resultat
%	a + b	heltal, dec.tal	heltal, dec.tal

'++' - operatorn (med flera)

I JavaScript, såväl som i C++ och Java, kan vi använda skrivsättet `i++` för att öka variabeln i:s värde med 1. Det är alltså detsamma som att skriva `i = i + 1;`. Det finns också det omvända, `i--`, vilket minskar i:s värde med ett. Som om inte detta vore nog finns det ännu fler operatorer av samma sort (det finns fler, men jag nöjer mig med dessa):

Symbol	Uttryck	Betyder
++	`i++;`	`i = i + 1;`
--	`i--;`	`i = i – 1;`
+=	`a += b;`	`a = a + b;`
-=	`a -= b;`	`a = a - b;`
*=	`a *= b;`	`a = a * b;`
/=	`a /= b;`	`a = a / b;`

Jag tycker inte de har så stor betydelse, man kan ju enkelt åstadkomma samma sak i alla fall. Men nu vet ni vad det betyder. Jag kommer att använda mig av dem i detta dokument.

Komparativa operatorer

Det finns ett antal fördefinierade operatorer för att jämföra värdet på två variabler. De är:

Symbol	Beskrivning
==	'är lika med'
!=	'är inte lika med'
>	'är större än'
>=	'är större än, eller lika med'
<	'är mindre än'
<=	'är mindre än, eller lika med'

Ulf Tornert
ulftt@gy.varmdo.se

Resultatet från en jämförelse med dessa operatorer är av typen boolean (logisk), och har värdet *true* eller *false*. Satsen 5 >= 3 ger därför resultatet *true*. Jag kan därför definiera en variabel `check`

```
var check = (5 >= 3);
```

som får värdet *true*. Man använder dessa ofta med kontroll- och loopstrukturer, vilka presenteras i nästa avsnitt.

Kontroll- och loopstrukturer

If … else – satsen

I alla programmeringsspråk återfinner du denna konstruktion som oftast syftar till att styra programmets flöde åt olika håll beroende på statusen hos en eller flera variabler. Grundsyntaxen för satsen är

```
if (testvillkor)
{
  // satser som körs om testvillkoret är uppfyllt
}
else
{
  // satser som körs om testvillkoret inte är uppfyllt
}
```

Ett testvillkor skulle kunna vara `if (x >= 10)`. Om x är större än, eller lika med 10, kommer satserna inom det första paret med klamrar (tecknen '{' och '}' – även kallade måsvingar) att köras. Om x är mindre än 10 körs satserna inom nästa par av klamrar. Klamrarna har en central roll då dessa särskiljer block med kod från varandra. Detta möjliggör för programmeraren att styra programmets flöde precis så som han/hon vill. Detta sätt att först testa ett villkor och sedan exekvera kodblock kan även utökas till *nästlade if…else*-satser:

```
if (testvillkor)
{
  kodblock..
}
else if (nästa testvillkor)
{
  kodblock..
}
else if (nästa testvillkor)
{
  kodblock..
} osv osv
```

Man skulle kunna tänka sig följande kod

```
if (x >= 10)
{
  document.write("Talet är större än, eller lika med 10.");
}
else if (x < 10)
{
  document.write("Talet är mindre än 10.");
}
else
```

```
{
  document.write(”Fel: Argumentet ej ett tal.”);
}
```

Här kommer det sista blocket endast att köras om de två komparativa testen ovan misslyckas, och det kan de bara göra om x inte är ett tal alls.

Villkorsoperatorn

Tänk dig att du vill sätta en variabel till ett av två värden beroende på om ett test resulterar i värdet sant eller falskt. Man kan med if…else-satsen tänka sig följande

```
var isNN;
if (navigator.appName == ”Netscape”)
  isNN = true;
else
  isNN = false;
```

Detta betyder att variabeln `isNN` får värdet `true` om webbläsaren är NN, och värdet `false` annars. Detta går att skriva på följande mycket bekväma sätt

```
var isNN = (navigator.appName == ”Netscape”) ? true : false;
```

Dvs. det vi tidigare skrev på fyra rader blir nu bara en rad. Detta skrivsätt används ganska flitigt vid exempelvis browserkontroller, och om ni någongång stöter på det vill jag att ni skall veta vad satsen betyder (syntaxen är ju något kryptisk).

for-slingan

Många gånger vill man göra samma sak ett upprepat antal gånger. Man brukar säga att man exekverar en *loop*, eller en *slinga*. Ett exempel kan vara att leta igenom en heltalsvektor efter tal som är större än 100, och sedan spara indexen till de talen. Dessa kan sedan bearbetas efter slingan på önskat sätt. Grundsyntaxen till for-slingan är

```
for (var i = start; i < stop; i++)
{
  // satser i slingan..
}
```

Du anger alltså ett startvärde för uppräkningsvariabeln `i`, en logisk test för när du skall stoppa och hur uppräkningsvariabeln skall uppdateras. Du måste ju inte deklarera uppräkningsvariabeln i *for*-slingan på det här sättet, du kan lika gärna göra det före slingan. Uttrycket `i++` betyder att `i`:s värde skall ökas med 1 varje gång slingan *körts igenom*. Om alltså startvärdet på `i` är 0 så kommer `i` att vara just 0 under hela första slingan. Därefter ökas `i` till 1 och slingan upprepas. Om stopvärdet är 10 kommer slingan att köras ända tills `i` är 9, men när `i`:s värde blir 10 avbryts hela *for*-slingan, eftersom `i < 10`, och skriptet fortsätter med nästa sats direkt efter slingan. Låt oss först skapa ett mycket enkelt (och lika meningslöst) exempel. Jag vill skriva texten ’Jag heter Kalle Katt!’ på skärmen tio gånger. Då skulle jag kunna använda följande for-slinga:

```
for (var i = 0; i < 10; i++)
{
  document.write(”Jag heter Kalle Katt!<BR>”);
}
```

Ulf Tornert
ulftt@gy.varmdo.se

Lägg märke till att jag sätter startvärdet till 0 och testuttrycket till `i < 10`. Det innebär att slingan kommer att köras fram till och med då i = 9, men inte 10! Från 0 till 9 är det 10 gånger, vilket stämmer med vad jag önskade mig. Lägg också märke till att jag lägger in en
-tagg innanför citationstecknen för att få en radbrytning vid varje utmatning.

while-slingan

Det är inte alltid man vet exakt hur många gånger man skall köra en slinga och då är for-slingan inte till mycket hjälp. Man kan då använda sig av while-slingan. Denna utvärderar ett logiskt test och kör sedan satserna i slingan om testvillkoret är uppfyllt. Grundsyntaxen är:

```
while (testvillkor)
{
  // satser i slingan..
}
```

Ett exempel skulle kunna vara om du har ett formulär med ett antal radioknappar (som alla har namnet `userChoice`) och du vill kolla vilken knapp en användare har valt. Lösningen kan se ut så här:

```
var i = 0;
while (!form.userChoice[i].checked)
{
  i++;
}
document.write("Du valde " + form.userChoice[i].value + ".");
```

Lägg märke till att jag definierar en uppräkningsvariabel *före* while-slingans ingång. Lägg också märke till testuttrycket som betyder 'om knappen `userChoice` nr i *inte* är nertryckt' så…
Det enda som görs i slingan är att `i`:s värde ökas med ett, detta för att testa om nästa knapp är nertryckt eller inte. Till sist vill jag göra dig uppmärksam på hur man konkatenerar (slår samman) uttryck i JavaScript. I satsen

```
document.write("Du valde " + form.userChoice[i].value + ".");
```

är 'Du valde ' en *sträng* och `form.userChoice[i].value` en *variabel* (som i och för sig hämtar en sträng). Jag lägger dessa till varandra genom att använda + tecknet. Strängar som vanligt inom citationstecken och variabler utan.

do-while slingan

Denna slinga används ungefär som while-slingan, dvs då man inte säkert vet hur många gånger en slinga skall köras. Det som skiljer dem åt är att do-while slingan först körs igenom en gång och utvärderar ett logiskt uttryck *efteråt*. Om detta är uppfyllt upprepas slingan. Grundsyntaxen för slingan är:

```
do
{
  // satser i slingan...
} while (testvillkor);
```

Ett problem där det krävs en slinga med okänt antal upprepningar brukar kunna lösas både med while- och do-while slingorna. Men ibland kan den ena kännas mer bekväm än den andra. Det står dig oftast fritt att välja en av dem.

Ulf Tornert
ulftt@gy.varmdo.se

switch-satsen

När man testar flera olika utfall av en variabel kan det vara fördelaktigt att använda switch-satsen. Man skulle kunna likna dess funktionalitet vid en *nästlad* if…else-sats. Grundsyntaxen för switch-satsen är:

```
switch (variabel)
{
  case utfall1:
    // satser…
    break;
  case utfall2:
    // satser…
    break;
  osv..
}
```

Den här konstruktionen tillåter programmeraren att låta skriptet ta *en av många* olika vägar beroende på statusen hos en variabel. Med `case` prövar du statusen hos variabeln och om den uppfyller testet kommer koden därefter att exekveras. Oftast vill man avbryta switch-satsen efter en sådan händelse och återlämna kontrollen till satserna efter *hela* switch-blocket. Det åstadkommer du med `break;`. Antag att du vill kontrollera om prickarna på en tärning representerar ett jämnt- eller ojämnt värde. Du skulle då kunna lagra värdet i en variabel – `nr` – och därefter låta denna passera följande test:

```
switch (nr)
{
  case 1: // Om nr är 1
    document.write(”Talet är ojämnt!”);
    break; // hoppa över resten av switch-blocket
  case 2:
    document.write(”Talet är jämnt!”);
    break;
  case 3:
    document.write(”Talet är ojämnt!”);
    break;
} och så vidare…
```

Detta sätt är nu ganska (läs: *mycket*) otympligt och det finns förstås många, mycket bättre sätt att lösa detta lilla problem på. Men för att demonstrera en finess med switch-satsen visar jag följande kod:

```
switch (nr)
  {
    case 1: case 3: case 5:
      document.write("Talet är ojämnt!");
      break;
    case 2: case 4: case 6:
      document.write("Talet är jämnt!");
      break;
  }
```

Man kan alltså stapla utfallen på varandra om dessa ändå skall få samma utgång i skriptet.

Ulf Tornert
ulftt@gy.varmdo.se

Logiska operatorer

Om du i en if…else-sats vill kontrollera värdet på två, eller flera, variabler *samtidigt* kan du använda vad som brukar kallas logiska (eller *booleska* operatorer). Anta att du vill kontrollera om en besökare har fyllt i sin e-post adress, och bockat för en checkbox i ett formulär. Anta att besökaren i checkboxen kan ange om han/hon vill bli kontaktad av dig. Det är ju bra om du får en e-post att skicka information till, och det är ju också bra om endast de som bockat för checkboxen blir kontaktad (för att slippa irritation). Du vill alltså kontrollera om textrutan för e-post **OCH** checkboxen för kontakt har korrekt indata. Anta att textrutan heter `email` och checkboxen `contact`. Båda finns i formuläret `userInfo`. Då kan du skriva så här:

```
if((document.userInfo.email.value != ””) && (document.contact.checked))
{
  // satser som körs om båda villkoren är uppfyllda
}
else
{
  // satser som körs annars…
}
```

I if-satsen ser du `&&`-operatorn, vilket motsvarar ordet **OCH**. Det betyder att villkoret till vänster-**OCH** till höger om operatorn måste vara uppfyllda samtidigt för att det övre satsblocket skall exekveras. Det vänstra villkoret betyder ”*om textrutan email **inte** är tom*”, det högra betyder ”*om checkboxen är förbockad*”. (Kom ihåg att operatorn `!=` betyder ”*är inte lika med*”). Den logiska kontrollen motsvarar därför nästan våra ambitioner. Det finns förstås en stor lucka i denna kod – kan du se vilken? Tänk om en besökare skriver in sin e-post adress som allan@@passagen.se. Vårt skript kommer att godta detta som godkänd e-post adress när den inte alls är det! Här måste vi gardera oss mot nonsens-indata. Mer om detta längre fram.

De logiska operatorerna vi har att tillgå är:

Operator	Namn	Exempel	Resultat
`&&`	`OCH`	`a && b`	`Boolean (`*`true`* `eller` *`false`*`)`
`||`	`ELLER`	`a || b`	`Boolean (`*`true`* `eller` *`false`*`)`
`!`	`INTE`	`!a`	`Boolean (`*`true`* `eller` *`false`*`)`

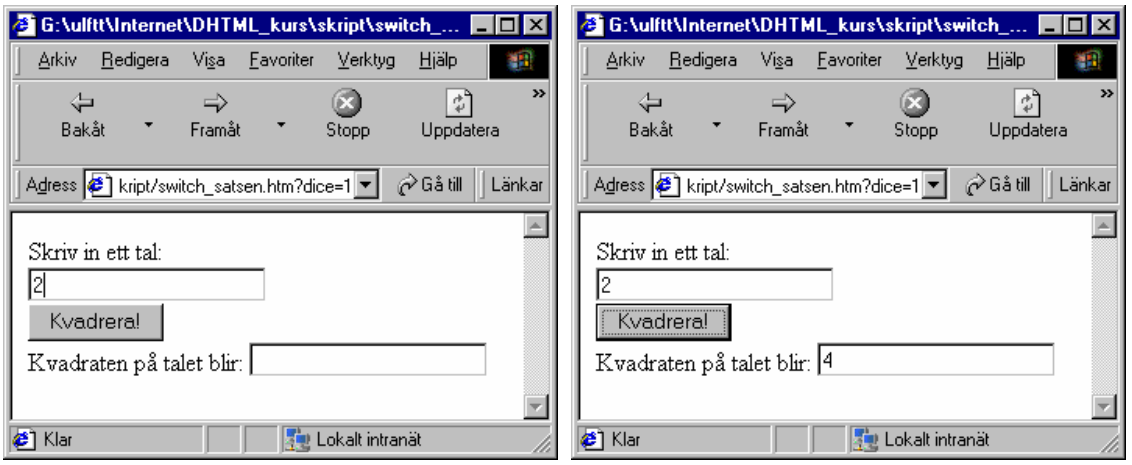
Det går också bra att stapla tester på varandra som i:

```
if ((a <= 5 && b > 3) || (a > 10 && b < 12))
```

vilket betyder ”*om (a <= 5 **OCH** b > 3) **ELLER** om (a > 10 **OCH** b < 12)*”. Parenteserna ger prioriteringsordning åt villkoren. Det här kommer att tolkas som ett logiskt test med ELLER-operatorn, där de båda villkoren till vänster- och höger i sin tur är logiska tester med OCH-operatorn (tillsammans med de komparativa operatorerna). Börjar det bli lite knepigt? Lugn – det ordnar sig!

Funktioner

Funktioner i JavaScript är de komponenter som utför de förändringar eller beräkningar som du vill ha i en given situation. Vi skall lära oss hur man skapar en funktion och hur man anropar den. Antag att du har ett formulär i en sida med en textruta där en användare kan skriva in ett tal. Du tar detta tal och beräknar kvadraten på det och skriver ut det i en annan textruta. Det skulle kunna se ut så här:



När jag skrivit in talet 2 i övre rutan trycker jag på knappen och får resultatet 4 i nedre rutan. Knappen måste alltså anropa en funktion, ta värdet från övre rutan, beräkna kvadraten och återge resultatet i nedre rutan. Hur gör man nu detta? Jag tror vi enklast tittar på koden till sidan:

```
<HTML>
<HEAD>
<TITLE></TITLE>

<SCRIPT LANGUAGE="JavaScript"><!--

function kvadrera()
{
   var nr;
   nr = document.forms[0].siffra.value;
   nr = nr * nr;
   document.forms[0].resultat.value = nr;
}

// --></SCRIPT>

</HEAD>
<BODY>

<form>
Skriv in ett tal:<BR>
<input type="text" name="siffra"><BR>
<input type="button" value="kvadrera!" onClick="kvadrera()"><BR>
Kvadraten på talet blir: <input type="text" name="resultat"><BR>
</form>

</BODY>
</HTML>
```

Funktionshuvudet

Vi börjar med funktionens huvud. En funktion måste börja med ordet `function` och därefter anger man funktionens namn med eventuella *parametrar* innanför parentesen. Med parametrar menas egentligen variabler man kan skicka med till funktionen, som därefter använder dessa för beräkningar. Om funktionen tar flera parametrar skall dessa skiljas åt med komma-tecken. Grundsyntaxen för ett funktionshuvud blir därför:

```
function funktionsNamn(parameter1, parameter2, osv)
```

Vår funktion har inga parametrar alls, eftersom parentesen är tom. Vi skall titta på hur man kan skicka med variabler till en funktion lite senare. Funktionshuvudet ser nu alltså ut så här:

```
function kvadrera()
```

Funktionskroppen

Efter huvudet kommer kroppen, och den skall omges med klamrar. Allt som står innanför dessa kommer att tillhöra funktionskroppen, och det är dessa satser som körs när funktionen anropas. I vårt fall innehåller kroppen:

```
{
  var nr;
  nr = document.forms[0].siffra.value;
  nr = nr * nr;
  document.forms[0].resultat.value = nr;
}
```

Den första raden innehåller en deklaration av variabeln `nr`. På rad två tilldelar jag denna värdet av det som står i textrutan med namnet `siffra`. På rad tre beräknar jag kvadraten på detta tal och slutligen på rad fyra presenterar jag resultatet i textrutan med namnet `resultat`. Så långt allt väl – men det finns fortfarande mycket att säga! Vad händer om någon skulle råka skriva in en bokstav istället för en siffra, eller om siffran inte är ett giltigt tal till ditt skript. Man måste oftast säkra upp skripten för den här typen av fel, men det tar vi senare!

Funktionsanropet

Jag har i det här fallet placerat funktionen mellan `<HEAD></HEAD>` taggarna. Det innebär att funktionen måste anropas på något sätt. I det här fallet löser jag detta med en s.k. *event-handler.* Det finns många sådana att tillgå och dessa avfyras när användaren gör något speciellt – exempelvis trycker på en knapp i ett formulär. Då avfyras (bl a) eventet *onClick* och jag kan fånga upp detta och se till att 'event-handlern' anropar min egen funktion. Jag ser alltså till att dirigera event-handlern *onClick* till min funktion med satsen

```
onClick="kvadrera()"
```

inuti `<input>` taggen.

Det finns ett annat mycket vanligt sätt att anropa en funktion. I NN och IE finns det en särskild URL som du kan använda – denna är `javascript:functionName()`. Här står `functionName()` för någon funktion som du själv har skapat. På det sättet kan man låta en länk anropa en funktion istället för att ladda ett nytt dokument. Det kan hända att funktionen du skapar faktiskt laddar ett nytt dokument, men den kan ju göra så mycket mer! För att låta en länk anropa funktionen `showInfo()` kan man skriva

```
<a href="javascript:showInfo()">Visa info</a>
```

där funktionen `showInfo()` är definierad någonstans i sidan (oftast brukar man definiera funktioner i `<HEAD>` sektionen).

Funktioner och lokala variabler

Variabeln `nr` i funktionen `kvadrera()` är vad man kallar en *lokal* variabel. Det innebär att endast funktionen själv har tillgång till den. Om jag i en *annan* funktion, eller *utanför* funktionen, försöker

att referera till variabeln `nr` kommer jag att få ett eventuellt skriptfel, eller logiska fel i skriptet. Däremot kommer alla funktioner att ha tillgång till variabler som deklareras som *globala*. Man brukar deklarera alla eventuella globala variabler innanför `<HEAD></HEAD>` taggarna. Om jag vill göra variabeln `nr` global i skriptet ovan kan jag möblera om lite:

```
<SCRIPT LANGUAGE="JavaScript"><!--

var nr;

function kvadrera()
{
   nr = document.forms[0].siffra.value;
   nr = nr * nr;
   document.forms[0].resultat.value = nr;
}

--></SCRIPT>
```

Nu är `nr` global och andra funktioner har också tillgång till den. Tag dock varning! Du kan få oavsiktliga förändringar av en global variabel, eftersom alla funktioner du skapar har tillgång till den. Man brukar försöka använda så få globala variabler som möjligt. Dessutom är det så att om du nu i funktionen `kvadrera()` skriver in satsen

```
var nr;
```

kommer denna att vara den variabel som de efterkommande satserna arbetar med, medan den globala variabeln lämnas orörd! Om du i en annan funktion behöver arbeta med detta värde måste du antingen arbeta med en global variabel, eller hellre skicka med variabeln till den funktion som skall arbeta med den. Det är därför viktigt att du tänker igenom från början hur du bör deklarera dina variabler!

Funktioner och parametrar

Jag visar nu kort hur man kan skicka med ett värde till funktionen. Betrakta koden nedan

```
<HTML>
<HEAD>
<TITLE></TITLE>

<SCRIPT LANGUAGE="JavaScript"><!--

function kvadrera(nr)
{
   nr = nr * nr;
   document.forms[0].resultat.value = nr;
}

--></SCRIPT>

</HEAD>
<BODY>

<form>
Skriv in ett tal:<BR>
<input type="text" name="siffra"><BR>
<input type="button" value="Kvadrera!"
onClick="kvadrera(this.form.siffra.value)"><BR>
```

```
Kvadraten på talet blir: <input type="text" name="resultat"><BR>
</form>

</BODY>
</HTML>
```

Funktionens huvud och kropp ser nu ut så här:

```
function kvadrera(nr)
{
  nr = nr * nr;
  document.forms[0].resultat.value = nr;
}
```

Nu innehåller funktionen en parameter som heter `nr`. Denna är inte deklarerad vare sig globalt, eller lokalt, men det blir en lokal variabel som skickas med anropet. Nu undersöker vi hur variabeln skickas. Jag har ändrat i event-handlern *onClick* så att denna anropar

```
kvadrera(this.form.siffra.value)
```

Det som står inne i parentesen är precis liktydigt med

```
document.forms[0].siffra.value.
```

och funktionen får därför – precis som förut – värdet i den första textrutan att arbeta med. Det betyder att `this.form` refererar till formuläret `document.forms[0]`. Nyckelordet `this` är lite speciellt och ibland mycket bekvämt att använda (det går att använda till mer). Kom ihåg att `forms[0]` refererar till det *första* formuläret i sidan. Och eftersom det bara finns ett enda formulär behöver jag inte vara särskilt noggrann med mina referenser. Jag skulle förstås kunna namnge formuläret och referera genom

```
document.formularNamn.siffra.value.
```

som du läste om under rubriken ’Referenser till namngivna objekt’.

Rekursiva funktionsanrop

Jag vill demonstrera en teknik till som används i samband med funktioner. Eftersom man kan programmera vilka satser som helst i en funktion, så borde man ju kunna låta en funktion anropa sig själv. Det kan man också – och det är dessutom användbart många gånger! Att låta en funktion anropa sig själv kallas för ett *rekursivt* funktionsanrop. Jag skall kort återge det kanske vanligast skolexemplet på sådana – nämligen i det fallet vi beräknar *fakulteten* på ett heltal. Fakulteten på ett heltal `n` definieras genom $n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-(n-2)) \cdot 1$ och skrivs på kortare form `n!`. Men vad betyder den här grekiskan då? Ja, vi tar talet `n` och multiplicer det med (n-1), sedan tar vi resultatet av den beräkningen och multiplicerar med (`n-2`) etc. etc. Vi avslutar repetitionen då vi kommer till talet 1. Ett exempel: $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$, vilket är lika med 120. Nu frågar man sig förstås om det finns något smart sätt att låta datorn räkna ut fakulteten för ett tal `n` (dvs `n!`). Vi måste förstås skicka med värdet från början så att funktionen vet vad den skall arbeta med, men sedan då? Kom ihåg: När man skickar med ett värde till en funktion så blir detta en *lokal* variabel som denna funktion arbetar med. Detta utnyttjar vi i följande exempel. Ta en titt på koden nedan (skriv gärna in den själv i en sida och kontrollera att den fungerar).

```
<HTML>
<HEAD>
```

Ulf Tornert
ulftt@gy.varmdo.se

```
<TITLE>Fakultet!</TITLE>

<SCRIPT LANGUAGE="JavaScript"><!--

function fak(n)
{
  if (n >= 2)
    return n * fak(n-1);
  else
    return 1;
}

function showRes()
{
  var n = parseInt(document.forms[0].siffra.value);
  // eventuell kontroll av indata
  document.forms[0].fakultet.value = fak(n);
}

// --></SCRIPT>

</HEAD>

<BODY>
<form>
Mata in ett heltal:<BR>
<input type="text" name="siffra"><input type="button" value="Beräkna"
onClick="showRes()"><BR>
Fakulteten på detta tal är:<BR>
<input type="text" name="fakultet">
</form>
</BODY>
</HTML>
```

Knappen i formuläret anropar funktionen `showRes()` via `onClick="showRes()"`. Denna funktion läser in värdet från textrutan `siffra` och gör ett heltal av detta med `parseInt()` (härefter bör man ha eventuell kontroll av indata, men för enkelhets skull utelämnar jag det här). Sedan ser jag till att textrutan `fakultet` får värdet av `fak(n)`. Det är detta funktionsanrop vi skall koncentrera oss på. Första kommentaren jag vill ge är på själva satsen

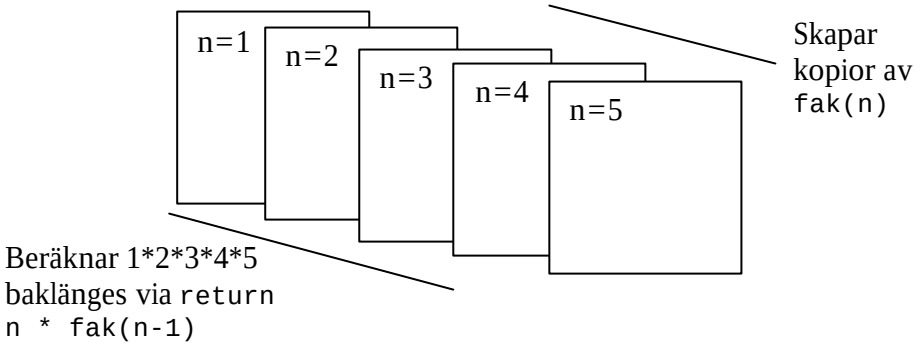
```
document.forms[0].fakultet.value = fak(n);
```

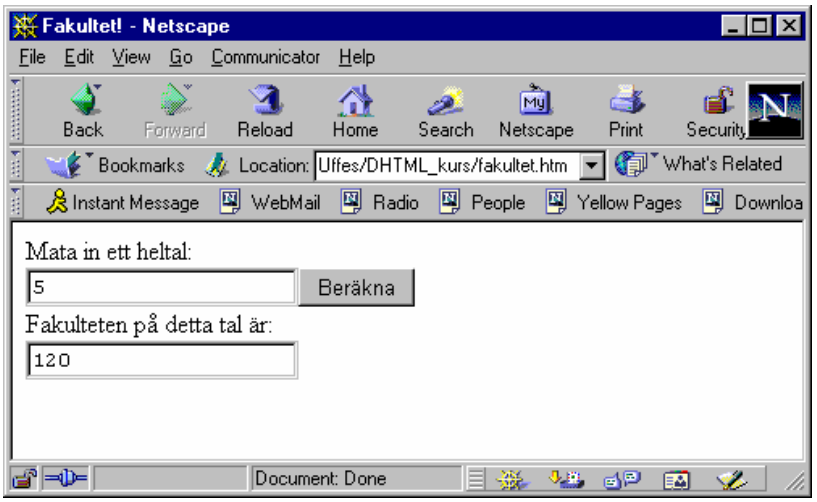
För att denna skall fungera måste ju funktionen `fak(n)` lämna ett resultat som kan skrivas in i textrutan `fakultet`. Det gör man med nyckelordet `return` inuti funktionen. Nu kommer vi till finnessen. Titta igen på funktionen `fak(n)`, anta att vi skrivit in talet 5 i textrutan `siffra`.

```
function fak(n)
{
  if (n >= 2)
    return n * fak(n-1);
  else
    return 1;
}
```

Den tar först talet 5 från textrutan, och testar om 5 är större eller lika med 2. Eftersom det är sant kommer funktionen att returnera `5 * fak(5-1)`, dvs vi kommer att få tillbaka 5 gånger 4. Men då har vi också anropat funktionen `fak(n)` två gånger, vilket innebär att vi har två kopior av

samma sort i minnet samtidigt. Det är dessutom så att de har *varsin* variabel med samma namn `n`. Men pga att denna är *lokal* kommer den första funktionen att arbeta med ett `n` som är värt 5, den andra med ett `n` som är värt 4 etc. Den kopia som arbetar med `n = 4` kommer ju att testa om 4 är större än 2. Detta är också sant, vilket innebär att vi kommer att få tillbaka 4 gånger 3 här. För varje gång vi anropar funktionen får vi ytterligare en kopia av funktionen i minnet, vilken har sitt eget `n` med ett värde som är ett steg mindre än den ”anropande” funktionen. Så håller det på tills `n` blir mindre än 2. Då kommer den sista kopian av `fak(n)` att returnera värdet 1. Det betyder att nivån ovan kan returnera 2 * 1 = 2, nivån ovan denna kan returnera 3 * 2 = 6, nivån därefter 4 * 6 = 24 och slutligen 5 * 24 = 120. Det betyder att beräkningarna inte startar förrän samtliga värden mellan 5 och 1 finns i minnet (i var sin kopia av `fak(n)`). Beräkningarna görs sedan baklänges, från den sist skapade kopian till den första. En liten principskiss över vad som sker i minnet kan se ut så här:





Det här är inte så lätt att förstå i sin helhet första gången, så bli inte missmodig om du tycker att det är svårt. Vi kommer inte att arbeta särskilt mycket med rekursion, men jag vill ändå belysa tekniken. Rekursiva funktionsanrop kan komma väl till pass vid exempelvis animering, eller tidräkning med JavaScript.

Events och eventhandlers

Med de grafiska gränssnitten i vår tids datorer och operativsystem ställs det stora krav på hur applikationer och operativsystem samarbetar för att fånga aktiviteten från den som kör applikationen. Om du som besökare på en webbsajt använder både mus och tangentbord för att navigera och fylla i formulär etc., vill du gärna att webbläsaren skall uppfatta alla kommandon du ger den för att besöket skall upplevas så smidigt som möjligt. Många objekt i JavaScript har s.k. *event-handlers* för att fånga upp aktiviteten från en besökare. Det finns en hel del olika event-

Ulf Tornert
ulftt@gy.varmdo.se

handlers för att ta hand om musklick, musrörelse, dubbelklick m.m. Vi har redan talat om *onClick* då vi kvadrerade ett tal från ett formulär. Namnet avslöjar ju vad det handlar om – när besökaren klickar med musen, avfyras ett *onClick*-event och detta fångas upp av event-handlern *onClick*. Event-handlern *onClick* är sedan under din kontroll och du kan dirigera den till att utföra vad som helst. Det fina med detta är att du som skriver JavaScript för en webbsida inte behöver ha en susning om hur denna process går till under ytan (vilket är rejält komplicerat). I JavaScript fångar vi *events* som Fantomen fångar skurkar – lätt och ledigt alltså! Jag kommer att vika ut några av de mest centrala objekten nedan och där listar jag också de eventuella event-handlers vi kan ha nytta av.

Objektet 'document'

Vi har redan tjuvtittat lite på detta objekt, och här kommer en mer utförlig (men inte komplett) beskrivning. Man brukar säga att ett objekt har *datamedlemmar, medlemsfunktioner* och *event-handlers.*

`document`-objektet:

Datamedlemmar	Medlemsfunktioner	Event-handlers
`anchors[]`	`captureEvents()`	`Inga!`
`applets[]`	`close()`	
`bgColor`	`open()`	
`cookie`	`releaseEvents()`	
`forms[]`	`write()`	
`images[]`	`writeln()`	
`lastModified`		
`layers[] (endast NN)`		
`links[]`		
`location`		
`title`		

Det är som sagt det här objektet som återspeglar vad som ryms i din sida. Objektet har faktiskt *ytterligare* någon information som inte direkt syns i webbläsaren, ett exempel är uppgifter om när filen senast ändrades (`document.lastModified`), sidans titel (innanför `<title></title>` taggarna) och lite till. Jag förklarar nedan lite kort vad var och en av de ovan nämnda attributen står för.

Datamedlemmar

anchors[], links[]

Jag har lite svårt att tänka mig vad denna datamedlem egentligen har för praktisk nytta, men jag tar med den ändå för att illustrera *ett* sätt att scrolla ett dokument via ett skript. I skriptet ingår sedan en del annat som kan vara värt en diskussion, bl.a. vad man kallar för *rekursiva* funktionsanrop. Det betyder egentligen att man inuti en funktion anropar samma funktion igen. Dessutom kommer modulo-operatorn väl till pass, samt en standardfunktion (`setTimeout`) som gör att vi kan pausa ett skript under en kort stund. Det jag skapar nedan är en helt fiktiv sida, där jag placerar ut ett antal ankare med radbrytningar emellan för att simulera sidans innehåll. I sidans huvud definierar jag den funktion som växlar dokumentets position och sist i kroppen anropar jag denna. Här följer koden för sidan:

```
<HTML>
<HEAD>
<TITLE>anchors[]</TITLE>
```

Ulf Tornert
ulftt@gy.varmdo.se

```
<SCRIPT LANGUAGE="JavaScript"><!--

var nr = 0;

function goNext()
{
  window.location.href = "#" + document.anchors[nr].name;
  nr = (nr+1) % document.anchors.length;
  t = setTimeout("goNext()", 5000);
}

// --></SCRIPT>

</HEAD>
<BODY>

<a name="first">Här börjar sidan.
<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR>
<a name="second">Här är andra ankaret.
<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR>
<a name="third">Här är tredje ankaret.
<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR>
<a name="fourth">Här är fjärde ankaret.
<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR>
<a name="fifth">Här är femte ankaret.
<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR>
<a name="bottom">Här är slutet på sidan.
<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR>
<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR>
<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR>

<script language="JavaScript"><!—-
  goNext();
// --></script>
</BODY>
</HTML>
```

Jag avslutar som sagt hela sidan med ett anrop till funktionen `goNext()`, denna sköter sedan allting helt på egen hand. Nu tar vi en titt på skriptet i sidhuvudet. För det första definierar jag en global variabel `nr`, vilken funktionen `goNext()` alltså får tillgång till. Jag sätter den till 0 från början, eftersom jag då kommer att starta sidan vid första ankaret (kom ihåg att `anchors[0]` refererar till det *första* ankaret i sidan). Den första raden i funktionskroppen är

```
window.location.href = "#" + document.anchors[nr].name;
```

Denna utnyttjar `window`-objektet för att komma åt adressfältet i browsern. Dvs `window.location` är inget annat än adressfältet i webbläsaren och `href` är en datamedlem till detta objekt, som endast är den textsträng som definierar den aktuella URL:en (Internetadressen). Nu sätter jag denna (via tilldelningsoperatorn '=') till `document.anchors[nr].name` inklusive tecknet '#'. Medlemmen `name` är namnet på ankaret, dvs `first` i `<a name="first">`. Ett ankare har, som du säkert kommer ihåg, en adress med sidans URL och tillägget `#ankare`. Ett exempel: `http://hem.passagen.se/ulftt/index.htm#intro`. Här tittar vi på sidan *index.htm* och sidans position är för tillfället vid ankaret *intro.* Eftersom den aktuella sidans adress är lagrad i browsern i sin helhet, kommer tilldelningen att resultera i hela adressen *plus* ankaret, dvs precis det vi vill ha! Lägg märke till hur man lägger till (konkatenerar) tecknet '#' till variabeln `document.anchors[nr].name`. Nästa rad har koden

Ulf Tornert
ulftt@gy.varmdo.se

```
nr = (nr+1) % document.anchors.length;
```

Här manipulerar `goNext()` med den globala variabeln `nr`. Den ökar först `nr` med ett och sedan dividerar den `nr` med antalet ankare i sidan, där divisionsresten behålls och denna läggs sedan till `nr`. Detta garanterar att vi alltid ökar `nr` med ett, och *om* `nr` skulle bli lika med antalet ankare i sidan kommer `nr` att få värdet 0 – dvs vi börjar då om från början med första ankaret! Tänk på att `document.anchors.length` kommer att få värdet 5, om det finns fem ankare i sidan. Men om du skriver `document.anchors[5].name` kommer du att försöka referera till ett ankare som inte finns (nämligen det ”*sjätte*”). Detta är anledningen till att modulo-operatorn är bra att använda här. Vid division med 5 ger den alltid ett resultat mellan 0 och 4, vilka samtliga ger relevanta referenser till ankare i vår sida! Sist i funktionskroppen kommer satsen

```
t = setTimeout("goNext()", 5000);
```

Denna utnyttjar funktionen `setTimeout()` för att göra ett rekursivt anrop till `goNext()` med fördröjningen 5000 millisekunder, dvs 5 sekunder. När fem sekunder har gått kommer `goNext()` att exekveras igen med ett nytt värde på `nr` så att nästa ankare läggs till URL:en! Frågan om skriptet är användbart eller inte, överlämnar jag åt din fantasi… Eftersom sidan självmant ändrar textsträngen i adressfältet (URL:en) kan det vara lite knepigt att komma ur slingan. Tryck på **CTRL-O** eller klicka på **File > Open**, för att öppna en ny URL.

Vektorn `links[]` har lite mer att ge. En länk är ju också något vidare än ett ankare. Man skulle kunna säga att objektet link har följande:

Datamedlemmar	Medlemsfunktioner	Event-handlers
`target`	`Inga!`	`onClick=`
`text`		`onDblClick=`
`x`		`onMouseDown=`
`y`		`onMouseOut=`
		`onMouseOver=`
		`onMouseUp=`

`target` är helt enkelt namnet på det fönster som länken kommer att laddas i, `text` är den länkade texten. Variablerna `x` och `y` är positionen för det övre vänstra hörnet i den tänkta box som håller länken. Dvs det är fullt möjligt att scrolla sidan till en exakt position där den aktuella länken finns. Objektet `window` har nämligen en `scrollTo(x, y)` funktion som tar en x- och y-koordinat och scrollar sidan till den positionen. Om du vill vara övertydlig med folk kan du därför hänvisa dem till en länk genom att ”slänga den i ansiktet på dem” genom

```
window.scrollTo(document.links[0].x, document.links[0].y);
```

Vad gäller event-handlers kan jag väl säga att den vanligaste är *onClick*. Denna går att använda på följande vis

```
<a href=”http://www.av.com” onClick=”check()”>Altavista</a>
```

Funktionen `check` utför någonting samtidigt som du laddar ned länken (eventuellt t.o.m. *innan*) i din browser. De andra event-hanterarna använder du på precis samma sätt.

applets[]

Java applets är små program som man kan köra i webbläsaren. Dessa har mycket större möjligheter än vad ett JavaScript har, och används ganska flitigt. Det positiva med det hela är att man inte behöver ha någon kunskap om Javaprogrammering för att dra nytta av dessa applets (eller *äpplen* som de också kallas). Det enda du behöver göra är att ladda ner den applet du vill köra och lägga den på din hårddisk (eller webserver) så att dina sidor kommer åt den. Sen bäddar du in appleten med `<APPLET></APPLET>` taggarna. När du gör detta kommer appleten att rulla igång. Men du kanske vill kunna kontrollera *exakt* när appleten skall dra igång – du kanske rent av vill att den skall stanna upp och vänta på input från ett formulär etc. etc. Hur gör man det? Svaret är att varje applet måste ha en *metod* för det! Du kan tänka dig en metod ungefär som en medlemsfunktion till ett objekt i JavaScript. Som exempel kan vi ta funktionen `write()` som hör till objektet `document`. Denna skriver ut en textsträng till en webbsida och syntaxen är

```
document.write(”Skriver en liten textsträng till webbsidan…”);
```

När du vet att det finns en sådan funktion till `document`-objektet är det ju bara att börja använda den. Hemligheten bakom att kunna styra dina applets är alltså att ta reda på om de har några användbara medlemsfunktioner – metoder. En bra applet har all denna dokumentation tillgänglig från den sajt du hämtar den ifrån – om den inte har det kan du få ordentligt svårt att hitta någon information alls. Låt säga att du hittar en applet som var 15:e minut ger dig en varning om att du är uppkopplad till Internet. Alla applets har två medlemsfunktioner som heter `start()` och `stop()`. Antag att appleten heter **ConnectionWarner.class** och att du lägger in den i din sida med

```
<applet code=”ConnectionWarner.class” name=”warning”></applet>
```

Då skulle du kunna stoppa den genom

```
<SCRIPT LANGUAGE=”JavaScript”>
<!—

document.warning.stop();

// -->
</SCRIPT>
```

Dvs du refererar till appleten precis som vilket annat objekt som helst, och sedan lägger du bara på metoden `stop()` med punktnotation som ”vanligt”. JavaScript kommer då att se till att rätt metod anropas till rätt applet (under förutsättning att du skriver rätt förstås). Applets är så vitt jag förstår harmlösa och de har i sin natur strikta begränsningar för vad de kan tillåtas göra med exempelvis dina filer på hårddisken o.dyl. Det är nämligen så att en applet aldrig har tillgång till sådant om du inte själv har godkänt det! Jag har ännu inte hört någon larmrapport, där någon stackare fått sin dator förstörd av en illvillig applet.

bgColor, lastModified och title

Variabeln `bgColor` används till att sätta bakgrundsfärgen till sidan. Detta är alltså den `bgColor` som ingår i `<BODY>` taggen! Du refererar till- och sätter bakgrundsfärgen med

```
document.bgColor = ”red”;
```

Antingen sätter du färgen – som här – med ett fördefinierat färgnamn, eller så använder du en färgkod. Glöm då inte ’#’ – tecknet som måste finnas med.

Ulf Tornert
ulftt@gy.varmdo.se

```
document.bgColor = ”#ff9966”;
```

Om du vill visa när dokumentet senast ändrades (och sparades) kan du lägga till koden

```
<script language="JavaScript"><!--

document.write("Detta dokument ändrades senast:<BR>");
document.write(document.lastModified);
document.close()

// --></script>
```

Detta sätt blir visserligen lite ’overkill’ – men det fungerar!

Som du säkert förstår kommer `title` att innehålla den textsträng som finns definierad i sidans `<TITLE>` tagg. Du kan extrahera den genom

```
var sidTitel = document.title;
```

location, cookie

Man kan fortfarande använda `document.location` för att ladda en ny URL, men rekommendationen är att istället använda `window.location.href` istället. Anledningen är att `document.location` kommer att tas bort ur DOM – åtminstone i NN. Du skulle alltså kunna ladda en ny URL genom

```
document.location = "nextpage.htm";
```

men det går också (bättre) att göra med

```
window.location.href = "nextpage.htm";
```

Cookies är som jag sa tidigare utanför kursen, men jag listar den här så att du vet att `cookie` är en datamedlem till `document` (vilket i och för sig är lite kryptiskt, eftersom en ”kaka” är en liten textfil på besökarens hårddisk).

forms[], images[] och layers[]

Vi har redan tidigare tagit en titt på `forms[]` och `images[]`. Vektorn `layers[]` har vi inte tittat på och jag skall kort nämna lite om den här. Men innan jag gör det repeterar jag vad vi gjorde under rubriken ’Referenser till objekt i sidan’. Då du bäddar in ett formulär i en sida kommer vektorn `forms[]` att innehålla en spegling av detta formulär. Anta att du har ett formulär som heter `IN_resurser` och att det finns en textruta där i som du kallar `favoritURL`. Om du sedan vill ta reda på vad det står i denna textruta kan du skriva

```
var URLen = document.IN_resurser.favoritURL.value;
```

Om det dessutom är det första formuläret i sidan kan du skriva

```
var URLen = document.forms[0].favoritURL.value;
```

Om det dessutom är så att textrutan `favoritURL` är först i det formuläret kan du skriva

```
var URLen = document.forms[0].elements[0].value;
```

Ulf Tornert
ulftt@gy.varmdo.se

där `elements[]` är en datamedlem till `form`-objektet, vilken innehåller en spegling av formulärets alla delar. Mer om detta under rubriken Objektet 'form'. Vektorn `images[]` innehåller en spegling av alla bilder i sidan. Låt säga att du har tre bilder i sidan, där den *tredje* bilden är inbäddad med koden

```
<IMG SRC="bild1.gif" NAME="nuna">
```

Denna kan du referera till på flera olika sätt

```
1) document.images["nuna"]
2) document.nuna
3) document.images[2]
```

Tänk på att alla vektorer i JavaScript har index 0 till första elementet. Det innebär förstås att den tredje bilden får indexet 2. Om du vill ändra på bildens källfil kan du göra det via `src`-attributet (och punktnotation med en av de tre ovanstående referenserna)

```
document.nuna.src = "bild2.gif";
```

Lager i NN hanteras med `<LAYER>` taggen. Vektorn `layers[]` kommer att innehålla en spegling av alla lager i sidan. Om du således skapar ett lager i sidan med

```
<HTML>
<HEAD>
  <TITLE></TITLE>
</HEAD>
<BODY>
<LAYER NAME="klass">
  …innehåll i lagret…
</LAYER>
</BODY>
</HTML>
```

kan du referera till lagret "klass" med `document.layers["klass"]`, `document.klass` eller `document.layer[0]` – precis som med en bild. Det finns dock en väldig skillnad på `images[]` och `layers[]` – alla lager innehåller nämligen i sin tur ett `document`-objekt! Det innebär att `document.layers[]` bara visar den första nivån av lager. Anta att du har en sida med följande kod

```
<HTML>
<HEAD>
  <TITLE></TITLE>
</HEAD>
<BODY>
<LAYER NAME="klass">
  …innehåll i lagret klass
    <LAYER NAME="person1">
    …innehåll i lagret person1
    </LAYER>
</LAYER>
</BODY>
</HTML>
```

För att komma åt lagret `person1` via ett skript måste du skriva

```
1) document.klass.document.person1
2) document.layers[0].document.layers[0]
```

eller något likvärdigt! Har du sedan ett lager i lagret `person1` måste du fylla på med ytterligare en `document`-referens. Mycket otympligt! Som vi senare i kursen kommer att se, så använder sig IE av en helt annan teknik för att referera till ett lager i sidan. Här kommer också problemen – för hur skriver man sidor som tar hänsyn till både NN och IE samtidigt? Båda webbläsarna är stora ute i världen och man kan inte exkludera den ena hur som helst. Åtminstone inte om man vill nå så många människor som möjligt!

Medlemsfunktioner

captureEvents(), releaseEvents()

Jag listar dem här för att kort påpeka att man kan ”fånga” en viss typ av event och peka samtliga på en enda funktion (om man så vill). Om man har en sida där samtliga kommandon av en sort – exempelvis *onMouseDown* – från besökaren skall resultera i en och samma utgång använder man `captureEvents()` med fördel. Om man sedan av någon anledning vill att eventen skall hanteras som vanligt ”frigör” man dem med `releaseEvents()`. Om du vill fånga alla *onMouseDown* och låta dem anropa funktionen `check()` kan du skriva

```
document.captureEvents(Event.MOUSEDOWN);
document.onMouseDown = check;
```

Lägg då märke till att det ska stå `Event.MOUSEDOWN` i parentesen till `captureEvents` och att parameterlistan helt skall uteslutas då `onMouseDown` pekas på funktionen `check()`. Kryptiskt! I exemplet ’Uppdatering av flera ramar’ under rubriken ’Några enkla exempel’ använder jag denna teknik. Den verkar dock strula med IE, så använd tekniken med stor försiktighet.

close(), open()

Du kommer förmodligen aldrig att använda medlemsfunktionen `open()`. Anledningen är att de funktioner som kräver att dokumentet ”är öppnat” i de flesta fall själva anropar denna explicit. Däremot kan du få användning av funktionen `close()`. Om du i din sida använder flera olika skript, som skriver till saker i den, kan det vara mycket viktigt att anropa `document.close()` vid slutet på varje skript sektion. Om man inte gör det kan man ibland få skriptfel när man egentligen inte skulle få det. Lägg särskilt märke till att det är mycket stor skillnad på funktionerna `document.open()` och `window.open()`. Den senare ska vi snart behandla.

write(), writeln()

Mycket användbara funktioner och vi har redan använt `write()`. Denna skriver ut den textsträng som anges i argumentet (innanför parentesen). Ett exempel kan vara

```
document.write(”Hello, world!<BR>”);
```

Funktionen `writeln()` har den fördelen att du kan utesluta `<BR>` taggen i satsen ovan – dvs den skriver automatiskt ut en sträng *inklusive* radbrytning.

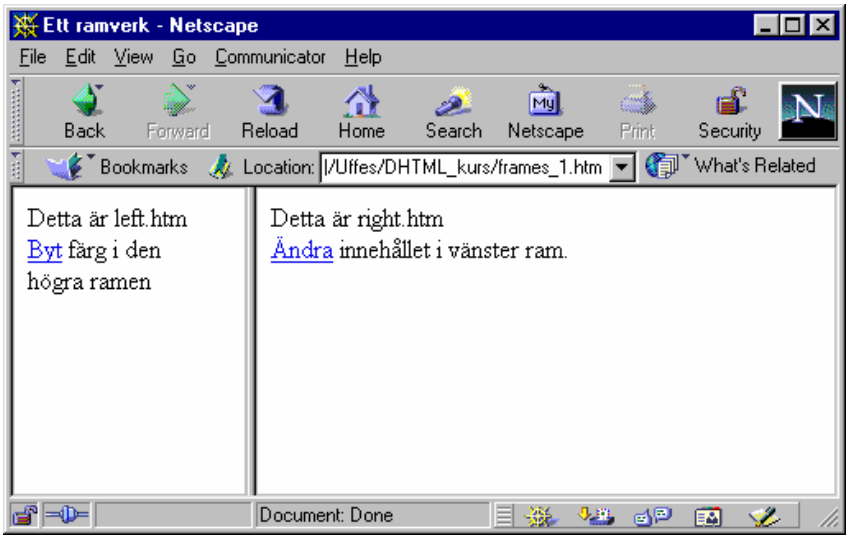
Objektet ’window’ och ramar

Förmodligen är `window`-objektet det största objektet i JavaScript. Du kan tänka dig detta objekt som hela webbläsarfönstret med navigeringsknappar, statusfält, adressfält, scrollbar etc. etc. Men det visar sig att en ram också är av samma typ, även om en ram naturligtvis inte har sin egen

uppsättning med navigeringsknappar, statusfält och så vidare. Detta faktum gör det hela lite förvirrande när man gör skript för sidor innehållande ramverk. Dessutom finns det flera olika fördefinierade namn på `window`-objekt – `top`, `self`, `parent` och `frames[]` – alla med olika effektiv innebörd, beroende på hur ditt ramverk är definierat (om du ens har något). Vi tittar på ett ramverk bestående av två kolumner:

```
<HTML>
<HEAD>
  <TITLE>Ett ramverk</TITLE>
</HEAD>
<FRAMESET COLS=”150, *”>
  <FRAME SRC=”left.htm” NAME=”nav”>
  <FRAME SRC=”right.htm” NAME=”main”>
</FRAMESET>
</HTML>
```

Den sidan skulle kunna se ut så här:



Detta enkla ramverk har tre olika `window`-objekt! Ett är `top` som innehåller ramverket (vilket i det här fallet är detsamma som `parent`), ett är `nav` som är den vänstra ramen och slutligen `main` som är den högra ramen. Hur skapar jag nu referenser mellan ramarna? Om du med ett skript i `left.htm` vill ändra bakgrundsfärgen i den högra ramen kan du göra detta genom

```
parent.main.document.bgColor = ”yellow”;
```

Tänk dig att du går från ramen `nav` *upp till* `top` och därefter *ner till* ramen `main` där du slutligen sätter bakgrundsfärgen. Om du å andra sidan från `right.htm` vill ändra innehållet i den vänstra ramen kan du göra det med

```
parent.nav.location.href = ”left2.htm”;
```

Med `parent` går man ett steg uppåt i hierarkin och sedan väljer man det fönster man vill manipulera med. I det här fallet går det alltså lika bra att byta ut `parent` mot `top`. Var dock något försiktig med att använda `top` hur som helst! Många gratis webbhotell, som Passagen, laddar din hemsida i ett ramverk, där `top` alltså kommer att vara något helt annat än vad du tänkte dig från början (detta går förstås att komma runt). Du skulle också kunna skriva

```
parent.frames[1].location.href = ”left2.htm”;
```

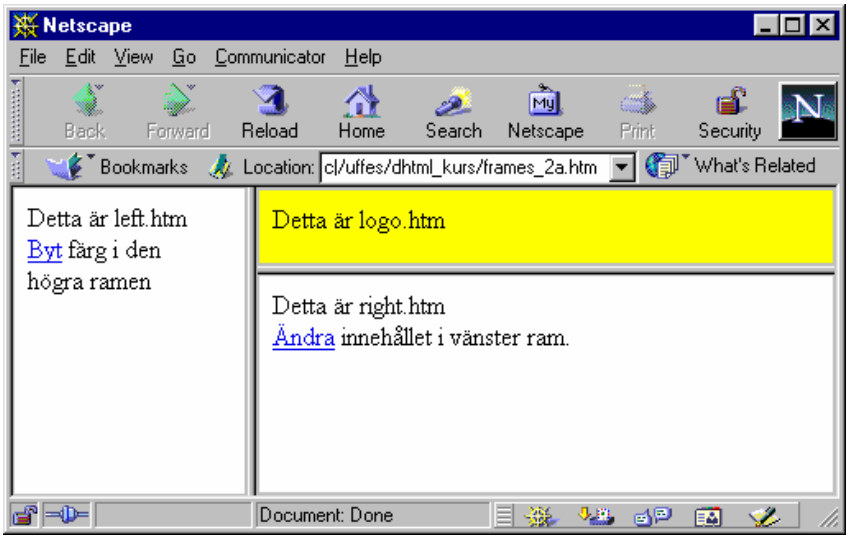
Nackdelen med detta är att referensen kanske bryter samman om du ändrar på ramlayouten. Det är oftast en bättre idé att använda de unika ramnamnen. Låt säga att du nu utökar ditt ramverk till följande

```
<!-- frames_2a.htm -->
<HTML>
<HEAD>
<TITLE></TITLE>
</HEAD>
<frameset cols="150, *">
   <frame src="left.htm" name="nav">
   <frame src="frames_2b.htm" name="main">
</frameset>
</HTML>
```

där `frames_2b.htm` i sin tur är ett ramverk med två ”rader” enligt

```
<HTML>
<HEAD>
<TITLE></TITLE>
</HEAD>
<frameset rows="50, *">
   <frame src="logo.htm" name="logo">
   <frame src="right.htm" name="info">
</frameset>
</HTML>
```

Ramverket skulle kunna se ut så här



Lägg noga märke till hur de olika ramarna får sina namn. Nu måste vi ändra på referenserna om vi vill utföra samma saker som förut. Eftersom fönstret `main` nu innehåller ett ramverk, i vilket ramen `info` är den vi vill åt, måste referensen i `left.htm` ändras till

```
parent.main.info.document.bgColor = "yellow";
```

om vi vill ändra bakgrundsfärgen i ramen `info` där `right.htm` är laddad. Om vi nu från `right.htm` vill ändra på innehållet i den vänstra ramen, måste vi tänka på att vi nu har en hierarki av `window`-objekt i tre generationer (man brukar tala om parent-child relationer). Längst upp har vi huvudfönstret (`top`), sedan kommer de två fönstrena `nav` och `main`, sedan kommer två fönster till i `main`-ramen, vilka har fått namnen `logo` och `info`. Om jag vill komma åt `nav` från `info` måste jag först gå två steg uppåt i hierarkin och sedan ett steg ner till `nav`! Det blir lite förvirrande, men med lite övning kommer det säkert att lossna. Så för att ändra innehållet i den vänstra ramen kan vi skriva

```
parent.parent.nav.location.href = "left2.htm";
```

där `parent` förstås refererar till `window`-objektet närmast ovan den egna ramen. Den observante frågar sig förstås varför jag ändrar på innehållet i vänstra ramen med den här knepiga konstruktionen – det med all rätt! Det vore ju mycket enklare att bara sätta ut en länk i `right.htm` med `target="nav"` och så var det klart. Men, då skulle jag ju inte kunna visa hur referenserna egentligen går till. Dessutom är det så att när jag skriver in länken i `right.htm` som

```
<a href="javascript:goURL()">Ändra</a> innehållet i vänster ram.
```

så har jag ju fullständig kontroll själv över vad funktionen `goURL()` skall utföra, eftersom jag definierar den själv. Den ser i nuläget ut så här

```
function goURL()
{
   parent.parent.nav.location.href = "left2.htm";
}
```

men om jag vill att den skall göra något mer än bara ladda en URL, så är det ju bara att mata in motsvarande satser i funktionen...

Nu när vi kan referera till ramar i olika sorters ramverk är det dags att ta oss an `window`-objektet i större utsträckning. Jag ger här en lista över vad objektet förfogar över, men listan är *inte* fullständig (om du verkligen vill veta måste du skaffa dig en heltäckande referens).

`window`-objektet:

Datamedlemmar	**Medlemsfunktioner**	**Event-handlers**
`closed`	`alert()`	`onBlur=`
`document`	`back()`	`onFocus=`
`frames[]`	`blur()`	`onLoad=`
`history`	`clearTimeout()`	`onUnload=`
`location`	`close()`	
`locationbar`	`confirm()`	
`menubar`	`forward()`	
`name`	`moveBy()`	
`opener`	`moveTo()`	
`parent`	`focus()`	
`personalbar`	`open()`	
`scrollbars`	`prompt()`	
`self`	`resizeBy()`	
`status`	`resizeTo()`	
`statusbar`	`scroll()`	
`toolbar`	`scrollBy()`	

top	scrollTo()	
	setTimeout()	

Det finns som sagt ett flertal variabler till som ingår i window-objektet, men för kursens räkning är det meningslöst att gå igenom precis allt.

Datamedlemmar

closed

Du kan med JavaScript öppna nya webbläsarfönster med navigeringsknappar, scrollbars etc. etc. Det kan hända att man via skript vill stänga ett sådant fönster igen. Datamedlemmen closed är av typen boolean och lämnar värdet true om fönstret du refererar till redan är stängt, antingen via skript eller av besökaren. Låt säga att du öppnat ett fönster med namnet newWin. Du skulle kunna pröva om fönstret är stängt i följande kontrollstruktur

```
if (newWin.closed)
{
  // satser om fönstret stängt
}
```

document

Som du säkert förstår är document en datamedlem till window-objektet. När man skriver document.bgColor = ”red” så är window-referensen underförstådd. Det betyder att webbläsaren tolkar satsen som window.document.bgColor = ”red”.

frames[], parent, self, top

Vi har redan tagit en titt på dessa. De är helt enkelt bara fördefinierade namn på fönster. Dessa namn är mycket användbara i ramverk.

location, history

Objektet location har vi också stiftat en del bekantskap med. Man kan säga att location är adressfältet i webbläsaren, men det innehåller mer än bara URL:en. Bland annat är det möjligt att extrahera de olika delarna av URL:en. Om du vill ändra dokumentet i ett fönster (eller en ram) kan du skriva

```
window.location.href = ”adressen till webbsidan”;
```

men det går faktiskt lika bra att skriva

```
window.location = ”adressen till webbsidan”;
```

Om du vill ändra dokumentet i en ram i ett ramverk kan du skriva

```
parent.frame[index].location.href = ”adressen till webbsidan”;
```

som förut, där frame[index] refererar till en särskild ram i ramverket. Objektet history har lite intressanta egenskaper. Det har några medlemsfunktioner och go() är en av dem. Funktionen go() tar ett positivt- eller negativt heltal som argument och stegar framåt- respektive bakåt så många steg i historien av alla de webbsidor du besökt som siffran motsvarar. Ett exempel

```
history.go(-1)
```

laddar in den sida som du besökte innan du kom fram till den nuvarande (som innehåller den här satsen). Denna funktion kan vara av intresse då du *vet* hur folk navigerar genom din sajt.

name, opener
Medlemmen `name` är helt enkelt bara namnet på det aktuella fönstret. Du kan plocka ut det genom exempelvis

```
var winName = window.name;
```

När du öppnar ett nytt fönster kommer detta att ha en medlem vars innehåll är namnet på det fönster som öppnade det. Det innebär att du kan stänga det första fönstret med det andra. Om från fönstret `main` skapar ett fönster med namnet `newWin`, så kan du ifrån `newWin` stänga `main` med antingen

```
1) main.close()
2) opener.close()
```

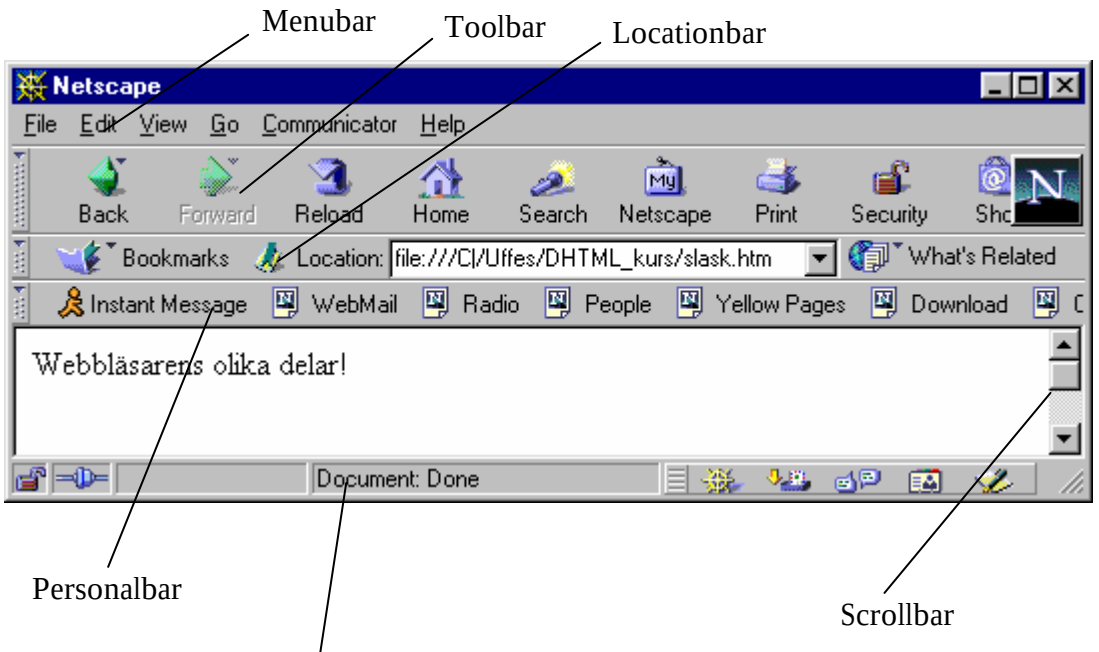
Resultatet blir att det fönster som skapade `newWin` stängs.

status
Detta är den textsträng som står längst ner i webbläsarens statusfält. Det är det lilla fält där man kan se vilken URL en länk pekar på. Man kan själv bestämma vad som ska stå i detta fält genom

```
window.status = ”Denna text hamnar i statusraden”;
```

Var dock försiktig med detta – det är inte särskilt snällt mot besökaren att lägga in massa onödiga kommentarer i statusfältet.

locationbar, menubar, personalbar, scrollbars, statusbar, toolbar
Dessa är de olika delarna av webbläsaren. Man kan öppna nya fönster och helt bestämma vilka delar som skall visas (om de alls ska visas). I bilden nedan visar jag vilken del som är vilken:



Ulf Tornert
ulftt@gy.varmdo.se

Jag visar under nästa avdelning hur man använder `window.open()` för att öppna nya fönster med exakt de egenskaper man vill ha.

Medlemsfunktioner

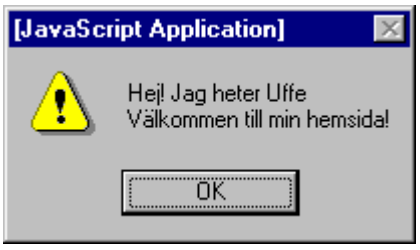
alert(), confirm(), prompt()

Det finns några standard funktioner till `window`-objektet, vilka ger författaren till sidan möjlighet att samverka med besökaren via Windows egna dialogboxar. Den första är en simpel varning av typen

Du kan lägga in en sådan vid en speciell händelse med satsen

```
alert("Ett ord av varning...");
```

En sådan speciell händelse kan ju vara att en besökare på en av dina sidor har skrivit in felaktiga data i ett formulär. Det kan då vara idé att låta denne få veta detta. Det är ganska bryskt och *kan* leda till en del irritation *om* den används för ofta. Det är nämligen så att dialogen *måste* avslutas innan resten av sidan kommer i funktion. Med andra låser man upp besökaren vid datorn, vilket inte är så bra. Det finns exempel på folk som besöker sidor mha makron i webbläsaren och sedan kollar själv när sidorna är 'cachade'. Att i det läget fastna i en sådan här dialogruta är inte populärt:



Däremot är `alert`-boxen mycket användbar under utvecklandet av dina skript. För att vara säker på att funktionsanrop funkar, och att exempelvis en variabel har ett speciellt värde i en given situation, är `alert`-boxen bra att använda. Sedan är det ju bara att ta bort alla sådana innan man publicerar sidan! Se bara till att få bort samtliga...

Nästa dialogruta är lite mer användbar (men var fortfarande mycket restriktiv). Vi kan kalla den `confirm`-boxen, eftersom medlemsfunktionen för att skapa en heter just `confirm()`. Den boxen har även en returtyp som är av typen `boolean`, dvs `true` eller `false`. Den kan se ut så här

Ulf Tornert
ulftt@gy.varmdo.se

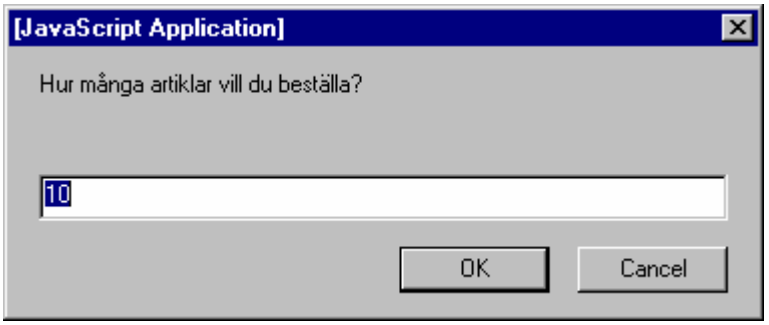
Koden för att skapa den är

```
var conf = confirm("Är du säker?");
```

När du trycker på `OK` kommer boxen att returnera `true` till variabeln `conf` (trycker du `Avbryt` blir värdet följdaktligen `false`). Det innebär att du kan använda den här boxen för att styra ditt skript med en `if...else`-sats.

```
<script language="JavaScript"><!--

if (confirm("Är du säker?"))
{
  alert("Du är tydligen helt övertygad!"); // Om OK
}
else
{
  alert("Nu måste du bestämma dig..."); // Om Avbryt
}

// --></script>

Här kommer innehållet i sidan...
```

Nästa dialogruta kallar jag `prompt`-boxen. Den är mer sofistikerad än `confirm`-boxen i det att du kan ställa en fråga och samtidigt ange ett default-värde som användaren antingen kan acceptera eller ändra. Returtypen är en sträng och boxen kan se ut så här



och koden för att skapa den är

```
var strang = prompt("Hur många artiklar vill du beställa?", "10");
```

Denna funktion tar två strängar som argument. Den första är det som blir frågan och den andra är default-värdet som visas i den vita textrutan. Alla dessa tre dialogrutor har den egenskapen att de måste klickas av innan resten av sidan kommer i funktion. Det betyder vanligtvis att de bör användas sparsamt om man vill att folk ska orka besöka ens sidor igen.

Ulf Tornert
ulftt@gy.varmdo.se

open(), close()

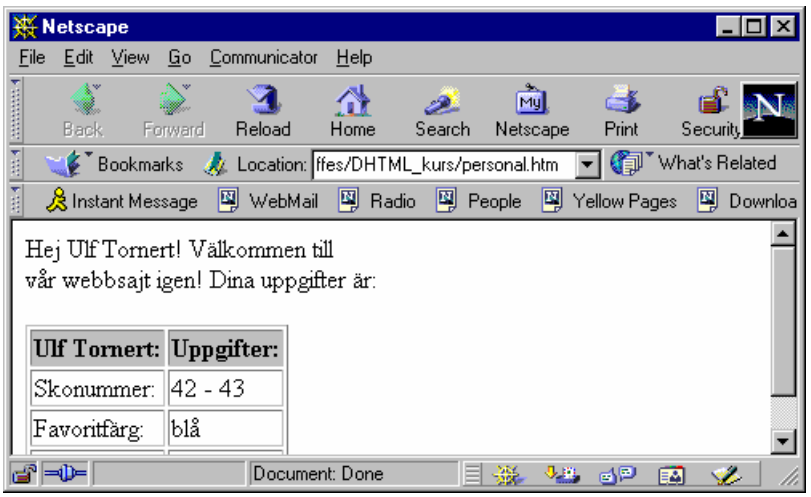
Nu ska vi titta på hur man från ett fönster kan öppna ett s.k. pop-up fönster. Dvs ett helt nytt webbläsarfönster, vilket författaren själv kan bestämma hur det skall se ut till storlek och innehåll. Då menar jag inte bara innehållet i själva sidkroppen (det är ju självklart) – utan hur själva fönstret ska se ut med navigeringsknappar, scrollbar etc. Standardsyntaxen för anropet är

```
open(”URL”, ”Namn”, ”övriga fönster egenskaper”)
```

`URL` är adressen till den sida du vill visa (adressen kan vara antingen absolut eller relativ). `Namn` är det alias det nya fönstret får. De övriga egenskaperna för fönstret har lite speciell syntax och funktionalitet, och jag kommer alldeles snart till det. Låt säga att du vill skapa ett pop-up fönster som laddar en personlig hälsning, innehållande personliga data från en databas i ett separat fönster, då kan du skriva

```
open("personal.htm", "");
```

Det fönstret skulle kunna se ut så här

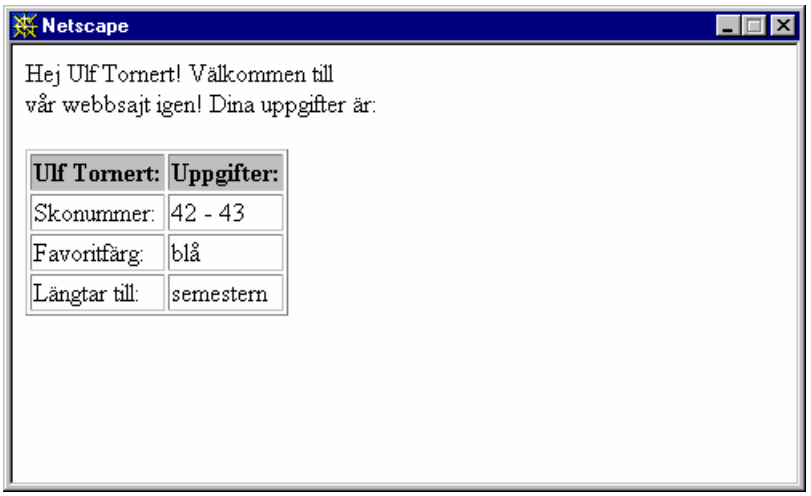


Det nya fönstret kommer att erhålla alla de egenskaper som webbläsaren skulle ha fått om du startat den från början (via Start-menyn). Dvs alla navigeringsknappar finns där, menyraden och scrollbaren likaså osv. Om du å andra sidan skriver

```
open("personal.htm", "", "height=200, width=300");
```

får du ett fönster som ser ut så här

Ulf Tornert
ulftt@gy.varmdo.se



Det betyder att fönstret får höjden 200 px och bredden 300 px – och viktigast av allt: **Alla** de andra egenskaperna utesluts! Så om du definierar en höjd och en bredd för fönstret, så måste du även ange vilka av de andra egenskaperna som skall finnas med. Om du då vill få med adressfältet, menyraden och statusfältet skriver du

```
open("personal.htm", "", "height=300, width=500, location=1, menubar=1,
status=1");
```

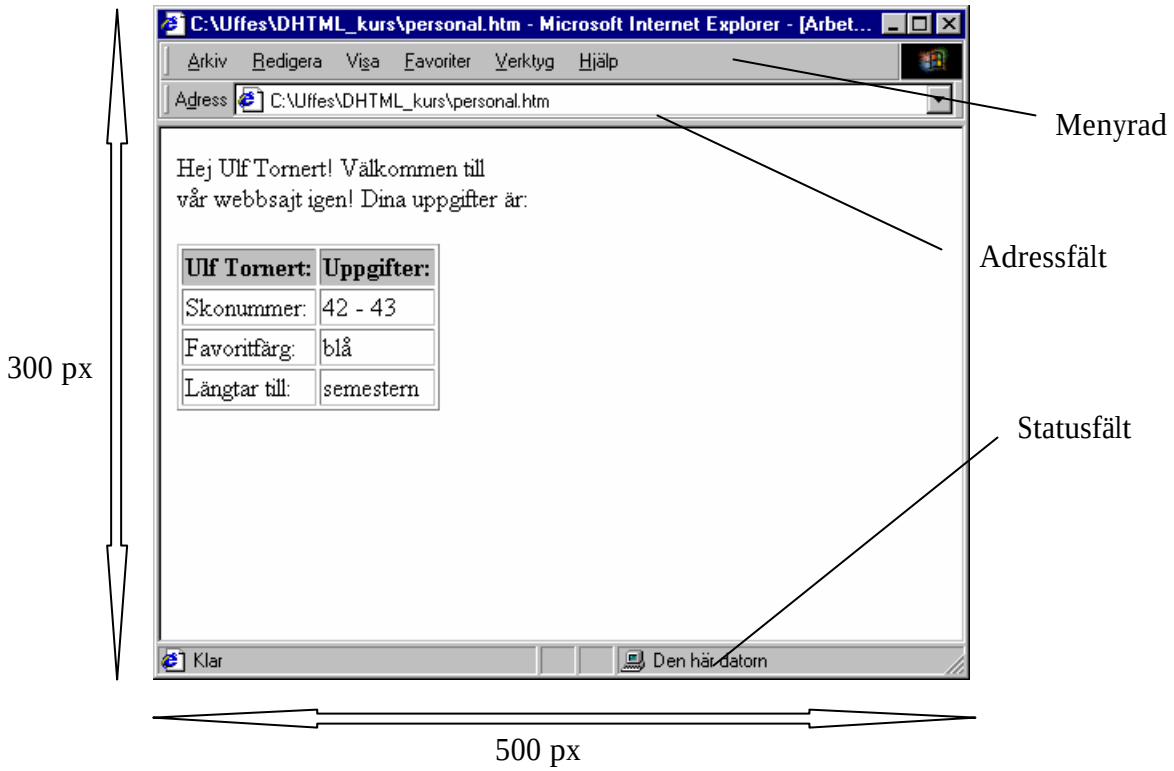
eller

```
open("personal.htm", "", "height=300, width=500, location=yes,
menubar=yes, status=yes");
```

Tänk på att detta fönster *inte* går att ändra storleken på. Om något inte får plats kommer besökaren inte att kunna se hela sidan! Man måste alltså ange explicit om fönstret skall vara ”*resizable*”. Lägg då till

`resizable=1` eller `resizable=yes`

i fönsteregenskaperna. Fönstret kommer nu att se ut så här



De egenskaper du kan påverka i funktionen `open()` är

Egenskap	Värde	Beskrivning
`toolbar`	`1` eller `yes`	Knappraden
`location`	`1` eller `yes`	Adressfältet
`status`	`1` eller `yes`	Statusfältet
`menubar`	`1` eller `yes`	Menyraden
`scrollbars`	`1` eller `yes`	Scrollbar
`resizable`	`1` eller `yes`	Storleken varierbar
`width`	pixlar	Fönsterbredd
`height`	pixlar	Fönsterhöjd

Om du vill stänga det nyss öppnade fönstret måste JavaScript veta hur fönstret skall nås från ett annat. Av den anledningen skapar man en variabel med ett passande namn. Denna variabel blir då av typen `window`, eftersom man skapar ett nytt fönster med `open()`. Det ser ut så här:

```
var newWin = open("personal.htm", "", "height=300, width=500,
location=yes, menubar=yes, status=yes");
```

Nu kan du använda ett formulär med en knapp, till vilken du använder *onClick* för att exekvera satsen

```
newWin.close();
```

Eftersom `newWin` är av typen `window` så har den också tillgång till funktionen `close()` och det är allt du behöver!

blur(), focus()

Dessa medlemsfunktioner används till att aktivera, eller inaktivera, det fönster som är aktuellt i en given situation. Med aktivt fönster menar jag det fönster som har den bekanta mörkblå namnlisten i Windows (de andra har en grå-grön namnlist), och som ligger längst fram på skärmen, närmast dina ögon. Om din sajt består av flera olika fönster där du presenterar lite olika typ av information, och där besökaren hänvisas fram och tillbaka, är det en fördel om besökaren själv slipper växla fönster mellan uppdateringarna. När du öppnar ett pop-up fönster, som i exemplet ovan, kommer detta att bli det aktiva fönstret automatiskt. Du kan sedan växla fritt mellan de olika fönstrena genom att från huvudfönstret exekvera satserna

```
newWin.blur(); eller newWin.focus();
```

eller att från pop-up fönstret skriva

```
opener.blur(); eller opener.focus();
```

Tänk på att om användaren har många fönster öppna samtidigt är det oklart vilket fönster som blir aktivt när du bara använder funktionen `blur()`. När du använder `focus()` så bestämmer man ju själv vad som blir aktivt – därför tycker jag att denna funktion är mer användbar.

back(), forward()

Dessa funktioner är författarens möjlighet att åstadkomma exakt detsamma som händer då en besökare trycker på back- eller forward-knapparna i menyraden till webbläsaren. För att testa funktionen kan du skriva in följande sats i en sida:

```
<script language="JavaScript"><!--
   var t = setTimeout("window.back()", 2000);
// --></script>
```

Funktionen `setTimeout()` kommer vi till lite senare i den här delen, och jag tar till den eftersom dokumentet endast kommer att återladda det närmast föregående dokumentet. Detta går mycket fort och man hinner inte se vad som händer. Ett annat sätt är ju förstås att skapa ett formulär med en knapp till vilken man anger `window.back()` till händelsen *onClick*. Funktionen `forward()` har samma syntax och funktionalitet (fast framåt i historien förstås). En liten varning kan vara på sin plats: Det finns två andra funktioner som heter exakt likadant, men som hör till `history`-objektet. Dessa växlar i den specifika historien för ett fönster *eller ram*. Funktionerna `window.back()` och `history.back()` är alltså inte desamma även om effekten ibland blir identisk. Detta beror ju på hur du har navigerat genom en sajt, och på om sajten är uppbyggd av ramar etc.

setTimeout(), clearTimeout()

Funktionen `setTimeout()` erbjuder ett sätt för författaren att tvinga JavaScript att vänta en liten stund innan nästa sats exekveras. Vi har redan använt denna metod två gånger nu – en gång i exemplet med `anchors[]` och en gång under föregående avsnitt. Grundsyntaxen för denna metod är

```
var t = setTimeout("funktionsnamn()", antal millisek);
```

Namnet på variabeln som refererar till timeouten är ju så klart valfri – här kallar jag den `t`. Funktionen du vill anropa skall anges inom citationstecken och efter kommatecknet anger du antalet ms (millisekunder) som du vill fördröja anropet med. Dock måste man se upp med hur

man använder metoden – detta eftersom JavaScript delar upp exekveringen i flera trådar samtidigt. Ett exempel:

```
<script language="JavaScript"><!--
   function doNothing(){} // En helt tom funktion – utför ingenting!
   var t = setTimeout("doNothing()", 5000);   // försöker pausa
   // här kommer satser som skall utföras efter pausen.
// --></script>
```

Först definierar vi en helt tom funktion som inte gör någonting, därefter anropar vi denna funktion via `var t = setTimeout("doNothing()", 5000);` Vi kan då luras att tro att funktionen `doNothing()` kommer att exekveras efter 5000 ms och resten av satserna därefter. Men vad som i själva verket händer är att medan `doNothing()` verkligen anropas efter 5000 ms, kommer resten av satserna att exekveras omedelbart! Och eftersom `doNothing()` verkligen inte utför någonting kommer vi aldrig att märka någon fördröjning alls! Hur gör man då? Svaret är oftast att använda `setTimeout()` med *rekursiva* funktionsanrop. Betrakta kodavsnittet nedan

```
function goNext()
{
   window.location.href = "#" + document.anchors[nr].name;
   nr = (nr+1) % document.anchors.length;
   t = setTimeout("goNext()", 5000);
}
```

Den sista raden i funktionen använder `setTimeout()` för att anropa *sig själv* med 5000 ms fördröjning. Eftersom det inte finns några fler satser att exekveras kommer inget att ske förrän `goNext()` exekveras igen – och det efter den önskade pausen. Detta är ofta en fruktbar metod (ibland den enda) för att skapa en paus i ett skript.

Funktionen `clearTimeout()` används i samarbete med `setTimeout()`. Man kan inaktivera en timeout genom att anropa `clearTimeout()` tillsammans med den variabel som refererar till timeouten. I exemplet ovan kommer variabeln `t` att referera till timeouten och jag kan nollställa denna med satsen

```
clearTimeout(t);
```

resizeTo(), resizeBy()

Dessa funktioner erbjuder dig en möjlighet att själv bestämma storleken på det webbläsarfönster som besökaren använder för att titta på din sida. Det kan ju vara användbart – även om jag personligen kan tycka att jag vill bestämma sånt själv. Om man nu nödvändigtvis vill forcera besökarens webbläsare att använda en viss storlek på fönstret så finns i alla fall möjligheten. Tänk dock på att delar av fönstret *kan* hamna utanför skärmen så att innehållet i din sida ändå inte syns – det skulle vara mycket pinsamt. Därför finns det även metoder för att flytta omkring hela fönstret på besökarens skrivbord. Mer om dessa senare. Grundsyntaxen för `resizeTo()` är

```
resizeTo(bredd px, höjd px);
```

Bredden och höjden skall anges i pixlar och utan citationstecken. Var mycket noga med att kontrollera effekten i både NN och IE, detta eftersom resultatet blir väldigt olika (ett ständigt återkommande problem för webbdesignern). Funktionen `resizeBy()` tar som argument det antal pixlar som det aktuella fönstret skall utökas med -–både horisontellt och vertikalt. Satsen

```
resizeBy(50, 50);
```

Ulf Tornert
ulftt@gy.varmdo.se

skulle alltså göra fönstret 50 pixlar större i både x- och y-led. Som jag sa: det kan vara nödvändigt för dig att explicit flytta fönstret så att hela innehållet garanterat syns. Ett illustrativt exempel på dessa metoder skulle kunna vara

```
<BODY>
<script language="JavaScript"><!--
   resizeTo(300, 300);
   var t = setTimeout("resizeBy(50, 50)", 5000);
// --></script>
Detta fönster har från början bredden 300 px och höjden 300 px.
</BODY>
```

Lägg märke till att fönstret först får storleken 300*300 px, därefter kommer texten i sidan. Slutligen kommer fönstret att utökas med 50 px i båda leden (tänk på att denna sats skall exekveras med 5000 ms fördröjning).

moveTo(), moveBy()

Dessa används nog i de flesta fall tillsammans med funktionerna i föregående avsnitt. De erbjuder möjligheten att placera ett fönster på valfri pixel från skärmens övre vänstra hörn (detta har koordinaterna (0, 0)). Syntaxen är ganska självklar

```
moveTo(0, 0);
```

Denna sats flyttar hela webbläsaren till skärmens övre vänstra hörn. Om jag sedan vill förflytta den kan jag göra det genom

```
moveBy(0, 50);
```

vilket skulle flytta fönstret 50 px i höjdled. Nu förfogar du alltså över möjligheten att förstora fönstret till – säg 800*600 px (vilket är en mycket vanlig skärmupplösning) – och därefter flytta fönstret så att allting syns! Effekten är alltså detsamma som om besökaren skulle välja att maximera sitt fönster själv.

scroll(), scrollTo(), scrollBy()

Det första jag vill säga är att `scroll()` bör skrotas till förmån för `scrollTo()`. Dels eftersom den senare är en yngre och bättre funktion som har precis samma effekt. Det sägs att `scroll()` inte fungerar som den ska. Dessa två kommer att rulla dokumentet till en given position, vilken du anger i pixlar. Grundsyntaxen är

```
scrollTo(x, y);
```

För att denna funktion skall vara riktigt användbar krävs nog att du väldigt väl känner till var objekt är placerade i ditt dokument. Om du förflyttar dig mellan ankare är det mycket bättre att använda `history`-objektet (som vi gjort tidigare). Funktionen `scrollBy(x, y)` tar också två argument i måttet pixlar, och sedan förflyttas positionen i fönstret med det angivna antalet *från den aktuella positionen*.

Eventhandlers

onLoad, onUnload

Dessa används ofta för att utföra grundläggande funktioner. När man kör ett skript måste ju det objekt man refererar till vara skapat – annars får man skriptfel. Objekt dyker upp i webbläsarens ”roadmap” (Document. Object Model) först då de påträffas i HTML-koden. Det kan vara bra att köra olika skript först *efter det att hela sidan har laddats* in i läsaren. På så vis minimerar man ju risken att referera felaktigt till ett objekt. Där kommer *onLoad* in i bilden. Betrakta koden nedan:

```
<BODY onLoad="init()">
Här kommer lite text...<BR>
</BODY>
```

Funktionen `init()` kommer inte att köras förrän *hela* dokumentet är laddat. Ibland mycket användbart. Lägg märke till att *onLoad* verkar höra till `document`-objektet, men sanningen är att den tillhör `window`-objektet (en ännu djupare sanning är att det inte spelar någon roll). Om du vill rensa bort eventuella pop-up fönster när en besökare lämnar din sajt kan du göra det med eventhandlern *onUnload*. För enkelhets skull har jag uteslutit funktionsdefinitionen som tar bort eventuella sådana fönster – det viktiga är att du ser hur man anropar funktionen. Betrakta koden

```
<BODY onLoad="init()" onUnload="rensa()">
Här kommer lite text...<BR>
</BODY>
```

Vi använder som förut *onLoad* för att anropa `init()`. Sedan får *onUnload* anropa `rensa()` – det är denna funktion som skall ta hand om eventuella pop-up fönster.

onBlur, onFocus

Dessa kan användas på samma sätt som de tidigare två event-hanterarna. Lägg dem i `<BODY>` taggen och ange den funktion du vill ska anropas vid de respektive händelserna. (Det kan fungera lite si och så har jag märkt, så tappa inte modet om du skulle få problem).

Objektet ’String’

En sträng i JavaScript är en serie tecken – en vektor med tecken – innanför citationstecken eller apostrofer. Det står dig fritt att välja vilka du vill arbeta med så länge du innesluter dina strängar i *ett par* av likadana avgränsare (citationstecken eller apostrofer). Lite längre fram kommer du att märka att jag blandar både och - men det har alldeles speciell användning. En sträng skulle kunna vara

```
var textStrang = ”En katt satt i en hatt”;
var textStrang = ’En katt satt i en hatt’;
var textStrang = ’<input type=”text” value=”Skriv här”>’
```

I det sista exemplet blandar jag de båda avgränsarna så att JavaScript kan tolka alltihop innanför apostroferna som en sträng. Orden *text* och *Skriv här* kommer också att tolkas som strängar, vilka senare kommer att tillhöra en textruta i ett formulär. Nedan visar jag några av de medlemsfunktioner som `string`-objektet förfogar över.

`string`-objektet:

Datamedlemmar	Medlemsfunktioner	Event-handlers

length	charAt()	Inga!
	indexOf()	
	lastIndexOf()	
	substring()	
	toLowerCase()	
	toUpperCase()	

Datamedlemmar

length

Den enda datamedlemmen som är av intresse för oss är `length`, vilken returnerar det heltal som motsvarar strängens antal tecken. Om vi skapar en sträng som ovan

```
var textStrang = ”En katt satt i en hatt”;
```

så kan jag undersöka hur många tecken den innehåller med satsen

```
var antal = textStrang.length;
```

Resultatet blir att `antal` får värdet 22, alla mellanslag räknas också som tecken. I de tillämpningar som vi senare skall göra kommer den här egenskapen hos `string`-objektet vara mycket användbar.

Medlemsfunktioner

charAt()

Denna funktion returnerar tecknet vid en viss position i den sträng som är aktuell. Kom dock ihåg att JavaScript indexerar positioner i en vektor – så ock i en sträng – från 0 och framåt. Grundsyntaxen för funktionen är

```
string.charAt(index);
```

Det allra första tecknet i strängen `textStrang` erhålls genom

```
textStrang.charAt(0); // returnerar ’E’
```

Vad händer nu om jag kombinerar denna funktion med medlemmen `length` på följande vis?

```
var tecken = textStrang.charAt(textStrang.length);
```

Detta kommer att översättas till

```
var tecken = textStrang.charAt(22);
```

men pga att JavaScript indexerar från 0 och framåt kommer detta att generera ett fel. Det visar sig att det sista tecknet i `textStrang` har index 21 – det finns inget tecken med index 22! Därför måste vi skriva så här

```
var tecken = name.charAt(name.length - 1); // returnerar ’t’
```

Nu kommer resultatet att stämma överens med vad vi önskar oss!

indexOf()

Denna funktion är något av en motsvarighet till `charAt()`. Funktionen returnerar indexet till det tecken som du anger i parentesen. Om tecknet inte finns returnerar funktionen värdet –1. Om det finns *flera* stycken kommer funktionen att returnera indexet till det *första* av dessa tecken. Du kan t.o.m. ange en sträng som argument till funktionen. Som ytterligare precision kan du ange var i strängen du skall börja leta. Grundsyntaxen för funktionen är

```
string.indexOf(”söksträng”, startvärde);
```

Några exempel:

```
var n = textStrang.indexOf("E"); // returnerar 0
var n = textStrang.indexOf("k"); // returnerar 3
var n = textStrang.indexOf("sat"); // returnerar 8
var n = textStrang.indexOf("en", 3); // returnerar 15, starta på index 3
var n = textStrang.indexOf("t", 3); // returnerar 5, starta på index 3
var n = textStrang.indexOf("t", 8); // returnerar 10, starta på index 8
var n = textStrang.indexOf("x"); // returnerar -1
var n = ”Lasse liten”.indexOf("x"); // returnerar -1
```

lastIndexOf()

Denna funktion fungerar som `indexOf()`, men returnerar indexet till det sista tecknet i den aktuella strängen som motsvarar det du stoppar in i parentesen. Grundsyntaxen är

```
string.lastIndexOf(”söksträng”, startvärde);
```

Några exempel:

```
var n = textStrang.lastIndexOf("E"); // returnerar 0
var n = textStrang.lastIndexOf("t"); // returnerar 21
```

substring()

Denna funktion låter dig kopiera delar av- eller en hel sträng. Grundsyntaxen för funktionen är

```
string.substring(start, stop);
```

Indexet `start` anger från vilket index du skall börja din kopiering, `stop` anger var kopieringen skall upphöra. Det senare är inte riktigt helt sant – kopieringen kommer att göras fram till det tecken som står *omedelbart före* det tecken som har index `stop`. Ett exempel:

```
var textStrang = "En katt satt i en hatt";
var subStrang = textStrang.substring(0, 4); // returnerar ”En k”
```

Bokstaven ’k’ har indexet 3, dvs omedelbart före 4 som anges i funktionsanropet ovan. Man kan också utesluta argumenten helt och hållet, JavaScript kommer då att tolka det som att hela strängen skall kopieras.

```
var textStrang = "En katt satt i en hatt";
var subStrang = textStrang.substring();
```

Satserna ovan resulterar i att `subStrang` innehåller ”En katt satt i en hatt”. Det knepigaste med funktionen `substring()` är att följande två satser är ekvivalenta:

```
var subStrang = textStrang.substring(2, 5);
var subStrang = textStrang.substring(5, 2);
```

När man har nytta av det är kanske en fråga för oraklet i Delphi?

toLowerCase(), toUpperCase()
Mycket användbara funktioner då man vill jämföra uttryck. Strängen 'JavaScript' är inte identisk med 'javascript' eftersom den förra innehåller versaler. Genom att först skapa strängar som innehåller endast gemener eller versaler får man säkrare jämförelser mellan strängar. Betrakta koden nedan

```
var strang = "Mitt namn är Bond, James Bond";
if (strang.toLowerCase() == "mitt namn är bond, james bond")
{
   document.write("Everything for a man on holiday!")
}
else
   document.write("You're a fake!")
```

Genom att anropa funktionen på strängen `strang` behöver man inte ta hänsyn till att vissa bokstäver är stora och att vissa är små. Detta är bekvämt, eftersom det oftast är bokstävernas kombination som är avgörande – inte dess format. Funktionen `toUpperCase()` returnerar förstås en sträng innehållande endast versaler.

Objektet 'Navigator'

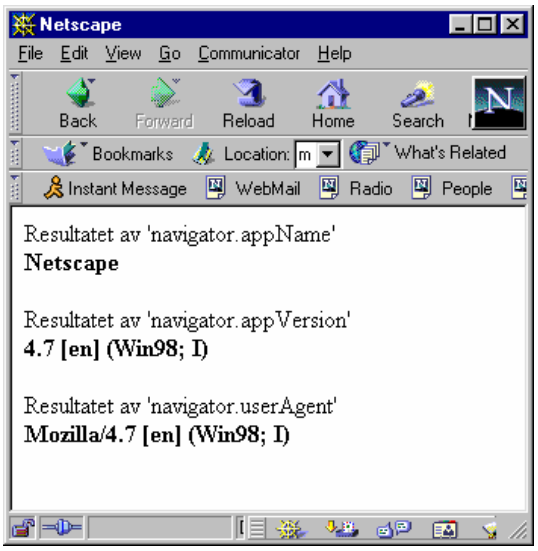
I JavaScript finns det speciella *kontextuella* variabler (eng. environment objects) som är mycket användbara då man som författare till en sida vill ta reda på exempelvis vilken typ av webbläsare en besökare använder. Informationen man kan få ut är ganska omfattande och inbegriper även sådant som vilka plug-ins som besökarens webbläsare för tillfället har installerade. Eftersom NN och IE fungerar lite olika kan det vara bra att i förhand veta vilken webbläsare som är aktuell. Dessutom är det så att de olika versionerna har väldigt olika stöd för JavaScript. Det är till exempel mycket stor skillnad på NN3 och NN4. Nedan ser du en lista över objektet `navigator` (ej komplett).

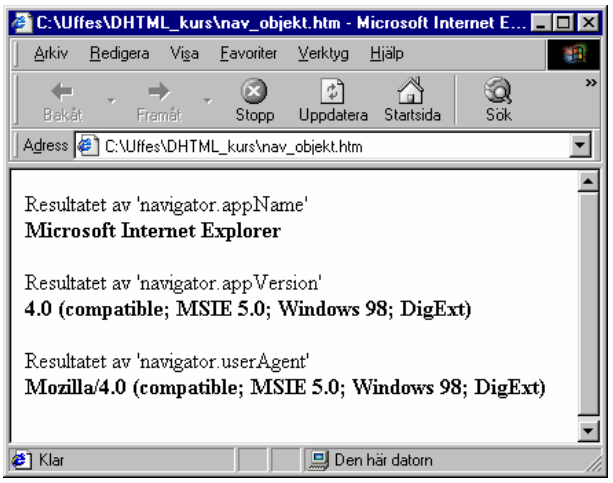
`navigator`-objektet

Datamedlemmar	Medlemsfunktioner	Event-handlers
`appName`	`javaEnabled()`	`Inga!`
`appVersion`		
`language`		
`platform`		
`userAgent`		

Datamedlemmar

appName, appVersion, userAgent
Alla dessa tre ger information om vilken webbläsare som laddat sidan (NN eller IE osv), vilket versionsnummer den har och lite till. Nedan visar jag resultatet av dem:





Det vi kommer att använda mest av allt är `appName` och `appVersion`, eftersom vi helst vill försäkra oss om att webbläsarna har versionsnummer 4 som minst (även om många har svårt att erkänna det, så är Internet Explorer en bättre webbläsare – åtminstone ur webbdesignerns perspektiv). Hur bär man sig då åt för att ta reda på lämplig information och hur styr man sedan skripten utifrån den givna situationen? Antag att vi vill ta reda på om en användare har antingen NN eller IE av version 4 eller högre, och utifrån det välja att köra ett skript som inte är kompatibelt med läsare av lägre versioner. Om läsaren har versionsnummer mindre än 4 skall något helt annat hända – kanske rent av ingenting om man är riktigt elak. Då kan man göra så här:

```
<HTML>
<HEAD>
<TITLE>En browser kontroll!</TITLE>

<script language="JavaScript"><!--
   var isNN4 = false;
   var isIE4 = false;
   function whichBrowser()
   {
      if (navigator.appName == 'Netscape' &&
   parseInt(navigator.appVersion) >= 4)
      {
         isNN4 = true;
         document.write("Gratulerar - NN4+ hanterar mina skript!");
      }
      else if (navigator.appName == 'Microsoft Internet Explorer' &&
                              parseInt(navigator.appVersion) >= 4)
      {
         isIE4 = true;
         document.write("Gratulerar - IE4+ hanterar mina skript!");
      }
      else
      {
         document.write("Din browser är av okänd typ - vänta medans du
         hänvisas vidare...");
         window.location.href = "other.htm";
      }
      document.close();
   }
```

```
// --></script>

</HEAD>
<BODY onLoad="whichBrowser()">
En liten browser kontroll...

</BODY>
</HTML>
```

Här använder jag *onLoad* för att starta en browser kontroll direkt efter att sidan har laddat. Det kan ju verka dumt eftersom mycket tid går till spillo *om* sidan innehåller mycket och *om* det skulle visa sig att besökaren har för gammal (eller okänd) browser! Lösningen ligger ju förstås i att göra index-sidan (första sidan) så liten och generell som möjligt – testa browsern samtidigt och ta nödvändiga åtgärder först då. Först när man *vet* att besökaren har en kompatibel browser ska man ladda ett större dokument med kompatibla skript – annars blir det sura miner. Ingen av oss gillar att ladda en sida i 15 sekunder för att få meddelandet att browsern är för gammal! Nu anropas `whichBrowser()` direkt efter att dokumentet laddats och där börjar jag med att definiera två variabler: `isNN4` och `isIE4`. Dessa kan användas senare i dokumentet eftersom de definieras som globala (utanför funktionen `whichBrowser()`). I funktionen kommer nu första testvillkoret

```
if (navigator.appName == 'Netscape' &&
            parseInt(navigator.appVersion) >= 4)
```

vilket betyder *'om det är NN **OCH** om versionsnumret är större än eller lika med 4 så…'*. Därefter sätter jag `isNN4` till `true`. Sedan kan jag ju göra vad som helst – för enkelhets skull skriver jag bara ut en enkel textrad. Om villkoren inte är uppfyllda kommer skriptet att köra igenom nästa test där vi undersöker om det är IE som laddats sidan

```
else if (navigator.appName == 'Microsoft Internet Explorer' &&
                            parseInt(navigator.appVersion) >= 4)
```

Om detta villkor är uppfyllt skrivs en likartad textrad ut (fast inte densamma) som ovan. Om inget av dessa villkor blir uppfyllda bryr vi oss inte om att undersöka fler varianter utan låter helt enkelt besökaren veta att dennes browser är av okänd typ och hänvisas därför till en annan sida (*other.htm*) som då lämpligtvis är uppbyggd av standard HTML som funkar i vilken browser som helst. Testvillkoren ovan går förstås att göra tillsammans med de stränghanteringsfunktioner vi gick igenom i föregående avsnitt. Ett sätt att testa om den aktuella browsern är IE är

```
if (navigator.userAgent.indexOf(”MSIE”) != -1 &&
            navigator.appVersion >= 4)
```

Eftersom funktionen `indexOf()` även tar strängar som argument, och eftersom Explorers `userAgent` alltid innehåller strängen 'MSIE' är detta också ett fungerande sätt. Kom ihåg att funktionen `indexOf()` returnerar heltalet –1 och den inte hittar söksträngen.

language, platform

Datamedlemmen `language` (NN specifik) returnerar en liten sträng bestående två tecken. Dessa är förkortningen till det språk som webbläsaren använder. Det finns en rad olika språk – några är

navigator.language	Språk
en	engelska
de	tyska
fr	franska

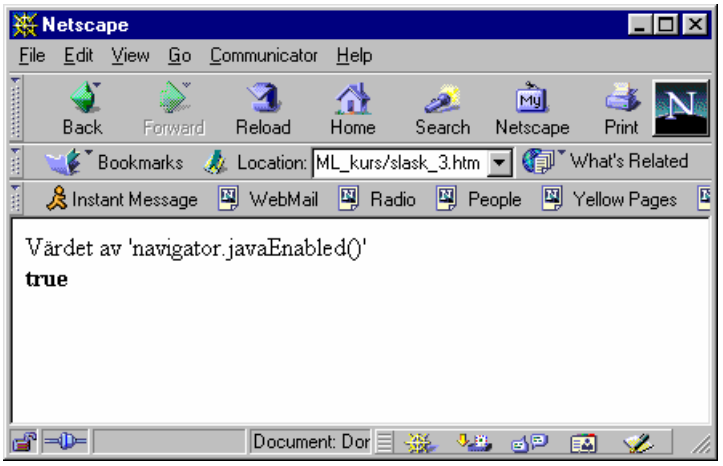
`it`	`italienska`
`ja`	`japanska`
`sv`	`svenska`

På detta sätt kan du ta reda på vilket språk din besökare talar (eller kan tala). Datamedlemmen `platform` fungerar i både NN och IE och returnerar en sträng, vilken är en förkortning för det operativsystem som webbläsaren är arbetar under.

Medlemsfunktioner

javaEnabled()

Som du säkert känner till kan man stänga av stödet för Java i både NN och IE. Om dina sidor innehåller Java-applets kan det vara av viss idé att kontrollera om Java-tolken är aktiverad eller inte (du kan då påminna- eller instruera besökaren att aktivera Java-tolken). Funktionen `javaEnabled()` returnerar alltså värdet `true` om Java är aktiverat och `false` om det omvända gäller. Exempel



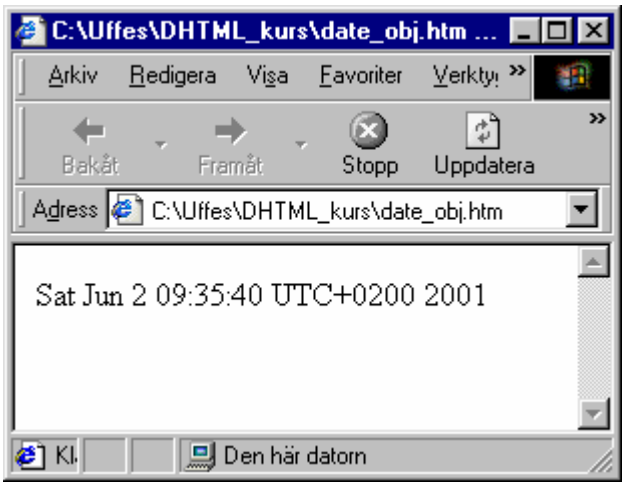
Objektet 'Date'

Säkert har du någon gång surfat in på en sajt och blivit hälsad med något i stil med "Välkommen till vår webbsida! Idag är det måndag den 4:e september och klockan är 10:07". För att en sådan hälsning skall fylla någon som helst funktion så måste ju datum och tid stämma överens för *varje*

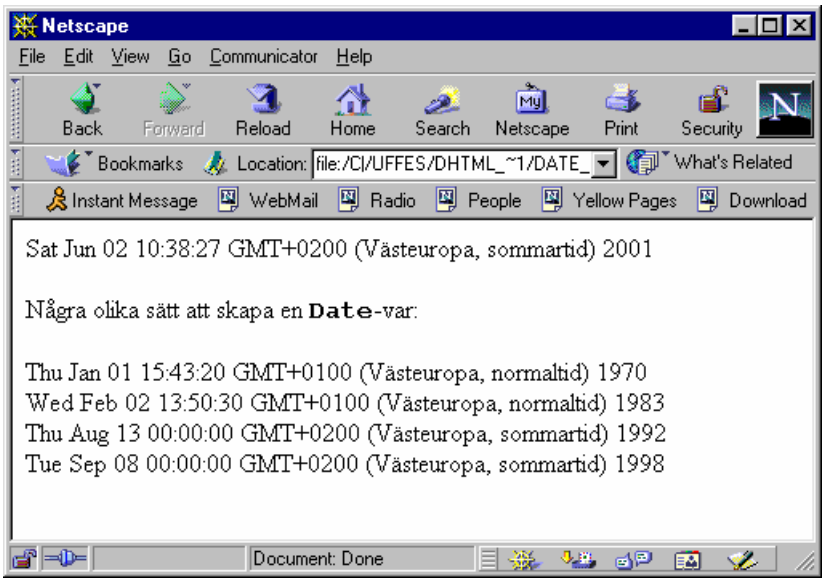
besökare *oavsett när* de laddar sidan. Hur gör man det? Svaret ligger i att man (exempelvis) med JavaScript kan ta ett 'snapshot' av systemklockan på besökarens dator och visa detta i sidan. Det finns då ett objekt som heter `Date`, vilket innehåller ett stort antal funktioner för att extrahera information ur en `Date`-variabel. I grunden kan man säga att en `Date`-variabel endast innehåller ett heltal. Detta heltal motsvarar det antal millisekunder som förflutit sedan den 1:a jan 1970, kl 00:00:00. Därefter använder man objektets medlemsfunktioner för att generera önskad information om månad, veckodag osv. Grundsyntaxen för att skapa en `Date`-variabel är

```
var idag = new Date();
document.write(idag);
```

Om jag därefter bara skriver ut variabeln `idag` i webbläsaren kan det se ut så här:



Inte särskilt snyggt! En hel del information – ja, men mycket otympligt att läsa. Och vad betyder 'UTC+0200'? Förkortningen UTC står för "Coordinated Universal Time" och är detsamma som GMT, vilket betyder "Greenwich Mean Time". I Greenwich, England finns den s.k. nollmeridianen och det är från denna man jämför all tid. När klockan är 12.00 i Greenwich, så är klockan ungefär 13.00 i Sverige, dvs det skiljer +1 timme. Att det sedan står +2 timmar ovan beror på att vi, när den här boken skrevs, har sommartid. Det finns ett flertal olika sätt att skapa en `Date`-variabel på, men oberoende på hur du skapar dem ser de likadana ut när du skriver ut dem som ovan.



Datum-variablerna ovan skapades med satserna:

```
dag2 = new Date(53000000);
dag3 = new Date("Feb 02, 1983 13:50:30");
dag4 = new Date("Aug 13, 1992");
dag5 = new Date(98,08,08);
```

Den som är mycket observant ser i den allra sista konstruktionen att `dag5` sätts till 98-08-08. Trots detta visar utskriften Thu Sep 08, … , 1998. Dvs månadsnumret 08 motsvarar september, vilket är den *nionde* månaden på året. Däremot är ”numret på dagen” 08, precis så som jag skrev in det. Här finns alla möjligheter att göra fel – håll tungan rätt i mun! Nedan visar jag de flesta av `Date`-objektets medlemsfunktioner.

`Date`-objektet:

Medlemsfunktioner	Värdeintervall	Beskrivning
`getTime()`	`positiva heltal`	`ms från 1/1/1970, 00:00:00`
`getYear()`	`70 - …`	`Årtal minus 1900`
`getMonth()`	`0 - 11`	`januari = 0, december = 11`
`getDate()`	`1 - 31`	`datum i månad`
`getDay()`	`0 - 6`	`söndag = 0, lördag = 6`
`getHours()`	`0 - 23`	`timme på dygnet`
`getMinutes()`	`0 - 59`	`minuter i den aktuella timmen`
`getSeconds()`	`0 - 59`	`sekunder i den aktuella min.`
`setTime()`	`positiva heltal`	`ms från 1/1/1970, 00:00:00`
`setYear()`	`70 - …`	`som ovan…`
`setMonth()`	`0 – 11`	`som ovan…`
`setDate()`	`1 – 31`	`som ovan…`
`setDay()`	`0 – 6`	`som ovan…`
`setHours()`	`0 – 23`	`som ovan…`
`setMinutes()`	`0 – 59`	`som ovan…`
`setSeconds()`	`0 – 59`	`som ovan…`
`getTimezoneOffset()`	`0 - …`	`minutförskj. från GMT/UTC`
`toGMTString()`		`ger sträng i std-format`

Ulf Tornert
ulftt@gy.varmdo.se

Nedan visar jag hur man skulle kunna fixa till en datum presentation mha ovan nämnda funktioner. Utskriften ser ut som nedan, därefter följer koden för sidan.


```
<HTML>
<HEAD>
<TITLE></TITLE>

<SCRIPT LANGUAGE="JavaScript"><!--

   var veckoDag = new Array(7);
   veckoDag[0] = "söndag";
   veckoDag[1] = "måndag";
   veckoDag[2] = "tisdag";
   veckoDag[3] = "onsdag";
   veckoDag[4] = "torsdag";
   veckoDag[5] = "fredag";
   veckoDag[6] = "lördag";

   function fixYear(ar)
   {
      if (ar < 100)
         return ”-” + ar;
      else
      {
         ar -= 100; // eftersom 2001 blir ’101’
         ar = "-0" + ar; // visa ’-01’ istället för ’1’
         return ar;
      }
   }

// -->
</SCRIPT>

</HEAD>
<BODY>
<script language="JavaScript"><!--

   var idag = new Date();
   dag = idag.getDay(); // ett tal från 0 till 6
   datum = idag.getDate(); // ett tal från 1 - 31
   manad = idag.getMonth(); // ett tal från 0 - 11
```

```
   ar = idag.getYear(); // ett tal från 70 och uppåt
   output = "Idag är det " + veckoDag[dag] + " " + datum + "/" + (manad + 1)
+ " " + fixYear(ar);
   document.write(output);
// --></script>
</BODY>
</HTML>
```

Lägg märke till hur jag sammanfogar (konkatenerar) en hel rad uttryck till variabeln `output`. Det känns ofta lättare att fixa utskriftssträngen för sig och därefter anropa `write()`, allt för att inte blanda ihop saker och ting. Det finns tillfällen då man med den har tekniken skriver ut delar av- eller rent av hela sidor. Om du vill skriva ut månaden med bokstäver får du skaffa dig en vektor med månadsnamnen i, på motsvarande sätt som jag gjort med veckodagarna ovan.

Objektet 'form'

Formulär erbjuder en besökare viss interaktivitet. Ett formulär kan fylla många olika funktioner – kontakt med företag, ansökningar om ex. nyhetsbrev eller medlemskap vid portaler, beställningar av varor, t.o.m. navigering på en sajt osv. När ett formulär har fyllts i och besökaren trycker på "Skicka"-knappen måste informationen skickas till webbservern. Den enklaste metoden för detta är att sätta `action`-attributet till `mailto:"e-postadress"` och `method` till `post`. För att resultatet skall vara lätt att läsa ska du även sätta `enctype` till `text/plain`. I det läget kommer det e-post program som besökaren använder att skicka innehållet i formuläret till den adress man anger efter `mailto:`. Detta har vissa uppenbara begränsningar – inte alla har ett e-post program installerat på sin dator, dessutom fungerar det ändå inte om besökaren uttryckligen inte har associerat sitt e-post program till att hantera formulär. NN och IE erbjuder förstås Messenger och Outlook Express, men besökaren måste likväl ha fyllt i viss nödvändig information för att det ska fungera. Webbdesignern är i det fallet helt utlämnad till användaren. Det finns då ett annat sätt, vilket fungerar helt oberoende av vad besökaren har för programvara på sin dator – CGI-skript. Förkortningen står för Common Gateway Interface och är små program på webbservern som anropas då formuläret skickas. Lite förenklat kan man säga att skriptet extraherar informationen från formuläret och skickar denna i ett mail till den adress du specificerar som formulärets mottagare. Ett CGI-skript är ofta skrivet i Perl och många sådana finns att hämta gratis på nätet. Eftersom de är speciella program måste formulären utformas för att passa ett särskilt skript. Den nödvändiga informationen finns nästan alltid tillgänglig på den sajt där du hittade skriptet. När du vet hur du skall utforma ditt formulär måste du förstås ladda upp skriptet på den webbserver där dina sidor ligger. Av tradition skall CGI-skript ligga i en katalog med namnet `cgi-bin`, eller liknande. Denna katalog måste ha särskilda rättigheter vilka din webbhost måste sätta. Pga att dessa särskilda rättigheter ger inte alla webbhosts möjligheten att ladda upp och köra vilka CGI-skript som helst, ofta får man inte köra CGI alls. Många har då löst problemet genom att erbjuda sina medlemmar ett fåtal standardskript som alla kan få köra. Du måste själv kontakta din host och fråga vilken policy de har, och om det finns standardskript som du kan få köra på din sajt (och i så fall vad du måste göra för att de ska fungera)! I den här delen kommer jag inte att behandla CGI drivna formulär, utan vi skall titta lite på vad man kan göra med JavaScript och formulär. Kom ihåg att JavaScript är klientbaserat och allt som sker utförs på besökarens dator, vilket underlättar trafiken på Internet. Det är en stor fördel (även om JavaScript har andra stora nackdelar).

När man skapar formulär med textrutor, textfält, select-boxar och knappar osv, så skapas motsvarande objekt i `forms[]`-vektorn, på samma vis som objekt skapas till `document`-objektet. Man brukar kalla detta för "road-maps". Referenser i formuläret görs på samma sätt som exempelvis referenser till `src`-attributet för en bild i ett dokument och man tar sig längre och

längre ner i hierarkin med punkter. Nu är formulärens ingående variabler också objekt – alla med sina speciella egenskaper. Därför kommer denna del att bli lite kortfattad. Om du verkligen vill ha bra koll på detta uppmuntrar jag dig att skaffa en heltäckande referens i ämnet. Men lite kommer du säkert lära dig redan nu! Nedan listar jag delar av `form`-objektet.

`form`-objektet:

Datamedlemmar	Medlemsfunktioner	Event-handlers
`action`	`reset()`	`onReset=`
`elements[]`	`submit()`	`onSubmit=`
`length`		
`method`		
`target`		

Datamedlemmar

action

Detta är precis detsamma som det du skriver in i `action`-attributet då du skapar formuläret. Du kan faktiskt både läsa och sätta detta värde på nytt via JavaScript. Man kan tänka sig att formulär skall behandlas av olika CGI-skript, beroende på hur en användare har fyllt i det. Du kanske t.o.m. vill ändra på adressen för `mailto:`. Det senare kan du göra genom

```
document.forms[0].action = ”mailto:min@adress.se”;
```

elements[], length

Datamedlemmen `elements[]` är en vektor med alla de objekt som ingår i formuläret. Om ett formulär innehåller två textfält och två knappar kommer `elements[]` att innehålla 4 objekt (element) – två textrutor och två knappar (inte helt överraskande). Om jag vet att det första elementet är en textruta kan jag få ut texten från den genom

```
var namn = document.forms[0].elements[0].value;
```

Tänk på att detta sätt att referera vilar på att du har *exakt* kontroll på hur ditt formulär är konstruerat. Detta eftersom objekten i `elements[]` dyker upp precis i den ordning du skriver in dem i din HTML-kod. Om du ändrar på formulärets design måste du också ändra på skriptet. Då kanske det är bättre att referera med unika namn istället. Detta sätt fungerar oavsett vilken ordning formulärobjekten hamnar. Om referensen ovan går till en textruta med namnet `fornamn`, i formuläret `besokare` kan du göra följande ekvivalenta referens

```
var namn = document.besokare.fornamn.value;
```

Om du sedan flyttar på denna textruta (inuti formuläret `besokare`) kommer referensen ändå alltid att fungera. Medlemmen `length` är ett positivt heltal och talar bara om *hur många* element som ingår i ett formulär.

method

Denna medlem motsvarar precis det som du anger till `method` då du skapar formuläret. Du kan via skript ändra den till antingen `get` eller `post`. Om du använder `mailto:` i `action`-attributet måste `method` vara satt till `post`.

target
Om du har ett CGI som skickar tillbaka en liten hälsning efter det att du sänt iväg formuläret, kan du peka denna hälsning till ett särkilt fönster eller ram. Det kan vara av intresse då du vill att besökaren fortfarande skall kunna se formulärets innehåll och läsa hälsningen samtidigt (den kan ju innehålla hänvisningar till formuläret).

Medlemsfunktioner

reset(), submit()
Funktionen `reset()` återställer formuläret till det innehåll formuläret hade då sidan först laddades. Om vissa textrutor och annat har default-värden kommer dessa att dyka upp igen. Denna metod anropas förstås om besökaren trycker på en knapp som är av *typen* `reset` – dvs om knappen definieras genom

```
<INPUT TYPE="reset" NAME="Reset" VALUE="Nollställ">
```

Det går också bra att anropa funktionen från något annat objekt – exempelvis en länk.

```
document.forms[0].reset();
```

Kom ihåg att JavaScript-funktioner har en särskild URL

```
<a href=”javascript:document.forms[0].reset()”>Nollställ</a>
```

Funktionen `submit()` fungerar likartat. Det är denna funktion som anropas då någon trycker på en knapp som är av *typen* `submit` som i

```
<INPUT TYPE="submit" NAME="Submit" VALUE="Skicka">
```

Men det går också bra att använda en länk (en vacker bild kanske?) som ”Skicka”-knapp också. Allt du behöver göra är att skriva in en javascript:URL och anropa `submit()` som i

```
<a href=”javascript:document.forms[0].submit()”><img
src=”skicka.gif”></a>
```

Tag dock ett ord av varning! Om du anropar en mer innehållsrik funktion där du dels vill skicka formuläret med `submit()` och dels göra lite annat – **gör då allt det andra först och anropa `submit()` sist av allt**! Annars kommer sändningen att avbrytas!

Event-handlers

onReset, onSubmit
Omedelbart efter det att en besökare tryckt på en `reset`-knapp kommer denna att avfyra en *onReset*-händelse. Denna går att fånga upp innan formuläret återställs om du i `<form>`-taggen anger vad som skall ske på en sådan händelse. Det kan ju vara så att en besökare av misstag råkat trycka på fel knapp. Betrakta följande kod

```
<form onReset=”return minReset()” onSubmit=”return minSubmit()”>
```

Om nu besökaren trycker på `reset`-knappen kommer `minReset()` att anropas. Den kan ju fråga om avsikten verkligen var att återställa hela formuläret. Om så var fallet skall man returnera `true`, om inte `false`. Det är därför det reserverade ordet `return` måste stå med – annars blir det fel. Samma förfarande gäller båda event-hanterarna. Dock är det så att *onSubmit* endast avfyras från

en riktig `submit`-knapp, dvs om du skickar formuläret via en bildlänk som ovan så kommer funktionen `minSubmit()` *inte* att anropas, utan formuläret skickas utan förvarning.

Textobjekt i formulär

Objektet 'text'

I de flesta formulär finns det någon form av textruta eller textfält. De olika typerna är

- textruta (`type=text`),
- lösenord (`type=password`),
- textfält (`type=textarea`) och
- dold text (`type=hidden`).

Alla dessa – utom dold text – har nästan samma uppsättning datamedlemmar, medlemsfunktioner och event-handlers. Det gör att jag nöjer mig med att lista dessa för textrutan (fast inte alla) och beskriver sedan kort skillnaden mellan denna och de andra.

`text`-objektet:

Datamedlemmar	Medlemsfunktioner	Event-handlers
`defaultValue`	`blur()`	`onBlur=`
`form`	`focus()`	`onChange=`
`name`	`select()`	`onFocus=`
`type`		`onSelect=`
`value`		

En textruta i ett formulär består av en enda rad med en bredd som du själv kan ställa in. Ett exempel:

```
<INPUT TYPE="text" NAME="fornamn" VALUE="Ulf" SIZE="30">
```

Datamedlemmar

defaultValue, value

I textrutan ovan kommer dess namn att vara `fornamn` och innehållet kommer att vara 'Ulf'. Detta är `defaultValue` för denna textruta. Oavsett vad en besökare skriver in i rutan kommer `defaultValue` alltid att förbli det som stod i `value`-attributet då formuläret skapades. Medlemmen `value` däremot står alltid för det som textrutan för tillfället innehåller. Om jag vill återställa denna enda textruta kan jag ju exekvera satsen

```
document.forms[0].fornamn.value = document.forms[0].fornamn.defaultValue;
```

name, type

Dessa två medlemmar kan vara bra för att försäkra sig om ett formulärobjekts typ och namn. Namnet `name` och typen `type` är värdet på de strängar du skrev in i `name`- och `type`-attributen då formuläret skapades. Du kan inte ändra på dessa i efterhand.

form

Här har vi en mycket intressant medlem. Med hjälp av denna kan vi bl a göra våra referenser rejält mycket kortare. Medlemmen `form` är ett alias för det *formulär* som textrutan är en del av.

Ulf Tornert
ulftt@gy.varmdo.se

Antag att jag har ett formulär i vilket jag förväntar mig att en besökare fyller i sitt förnamn. När denna gör det vill jag kontrollera att namnet inte innehåller skräpinformation. För att göra detta anropar jag en funktion via händelsehanteraren *onChange*. Till funktionen skickar jag med en referens till det aktuella formuläret (ännu enklare är att skicka bara den aktuella textrutan). Det ger mig två vinster – dels kan jag använda funktionen till vilket formulär som helst, eftersom referensen skickas med till funktionen, dels blir referensen mycket kortare. Betrakta koden nedan (funktionaliteten är övernitisk, men jag vill demonstrera tekniken):

```
<HTML>
<HEAD>
<TITLE>Lite formulärkontroll med JavaScript.</TITLE>

<SCRIPT LANGUAGE="JavaScript"><!--

function checkIt(form)
{
   if (form.elements[0].value != "")
   {
      var svar = confirm('Är ditt namn ' + form.elements[0].value);
      if (!svar)
      {
         var namnet = prompt("Ditt förnamn?");
         form.elements[0].value = namnet;
      }
   }
}

// -->
</SCRIPT>

</HEAD>
<BODY>

<FORM name="besokare">
<BR><INPUT TYPE="text" NAME="fornamn" SIZE="30" MAXLENGTH="30"
onChange="checkIt(this.form)"><BR>
<BR><INPUT TYPE="text" NAME="efternamn" SIZE="30" MAXLENGTH="30"><BR>
<BR><INPUT TYPE="reset" NAME="Reset" VALUE="Återställ">
<INPUT TYPE="submit" NAME="Submit" VALUE="Skicka">
</FORM>

<FORM name="besokare2">
<BR><INPUT TYPE="text" NAME="fornamn" SIZE="30" MAXLENGTH="30"
onChange="checkIt(this.form)"><BR>
<BR><INPUT TYPE="text" NAME="efternamn" SIZE="30" MAXLENGTH="30"><BR>
<BR><INPUT TYPE="reset" NAME="Reset" VALUE="Återställ">
<INPUT TYPE="submit" NAME="Submit" VALUE="Skicka">
</FORM>

</BODY>
</HTML>
```

Här har jag två formulär som ser exakt likadana ut, men för JavaScript är de verkligen olika. Referensen till de är

1. `document.forms[0]` eller `document.besokare`
2. `document.forms[1]` eller `document.besokare2`

Ulf Tornert
ulftt@gy.varmdo.se

När jag via händelsehanteraren *onChange* anropar `checkIt(this.form)` kommer argumentet inuti parentesen att motsvara ovanstående referenser beroende på vilken textruta det är jag ändrar i. Om jag har ändrat i rutan så att den inte längre är tom kommer jag att få en förfrågan om namnet stämmer, om det inte gör det får jag en ny chans att skriva i mitt namn varpå namnet i formuläret också ändras, om det stämmer händer ingenting. Ta inte det här exemplet på allvar – jag vill bara visa hur man *kan* göra. Som jag sade innan – det hade varit mycket enklare att skriva endast `this` i funktionsanropet, men den referensen går inte till formuläret, utan endast till den aktuella textrutan. Tanken var att jag skulle visa att `this.form` är en referens till hela det formulär som textrutan är en del av.

Medlemsfunktioner

blur(), focus(), select()

Funktionen `blur()` skulle kunna ges ett annat namn – `unfocus()`. Funktionen tar nämligen fokus från, och avmarkerar en textruta vars textinnehåll eventuellt är markerat. Normalt sett sker en sådan avmarkering när besökaren klickar utanför den aktuella textrutan, eller tabbar sig ur den (dvs trycker på TAB-tangenten). Men du kan också erhålla samma resultat via ett skript. Den mest omedelbara tillämpningen borde vara då man i ett formulär visar någon text i en textruta som besökaren inte ska kunna ändra. I `<input>`-taggen kan du då lägga till följande

```
onFocus=”this.blur()”
```

Så fort besökaren klickar i textrutan ”tappar” den fokus direkt! Funktionen `select()` å andra sidan åstadkommer nästan det direkt omvända – den ger en textruta fokus. Med detta menas att markören börjar blinka i rutan och det blir möjligt för en besökare att skriva in något i den. För att förenkla för besökaren kan man även direkt efter `focus()` anropa `select()`, vilken markerar hela den aktuella texten som finns i rutan. Det är då bara att skriva över det nya innehållet utan att först radera. Det senare skulle man kunna åstadkomma med

```
onFocus = ”this.select()”
```

Event-handlers

onBlur, onFocus, onSelect

Dessa tre händelsehanterare kräver noggrann eftertanke innan de används – särskilt om fler än en skall användas till samma textruta. Anledningen är att vissa händelser genererar flera händelser nästan samtidigt, vilket innebär att deras respektive händelsehanterare avfyras mer eller mindre samtidigt. Det är då svårt att göra felfria skript för att bemästra sådana händelser. Ett exempel är att *onFocus* avfyras direkt före *onSelect* om textrutan inte hade fokus före det att besökaren markerar texten i den. Hur tacklar man det? Vilken hanterare skall man välja att utgå ifrån? Riskerar man att få multipla funktionsanrop från olika händelser? Här måste du tänka dig för och testa noggrannt innan du publicerar dina formulär på webben! I kortaste laget kan man säga att *onBlur* avfyras då en besökare (efter att textrutan haft fokus) antingen klickar utanför rutan, eller tabbar sig ur den. Hanteraren *onFocus* avfyras då någon klickar i en ruta, eller tabbar sig in i den. Slutligen *onSelect* avfyras då någon markerar den text som finns i rutan.

onChange

Detta är den händelsehanterare jag anser vara mest lämpad för formulärkontroll. När en besökare har ändrat ett värde i en textruta och sedan antingen klickar sig ur den, eller tabbar sig ur den, avfyras händelsen *onChange.* När detta händer kan man omedelbart kontrollera om indata är korrekt och om det inte är det kan man be besökaren att ändra detta direkt. Det har den fördelen

att besökaren har i färskt minne vad han/hon just skrev in, eller skulle ha skrivit in. Sedan finns det ju inget som hindrar dig från att göra en slutlig kontroll på *onSubmit* om du vill...

Objektet 'password'

Detta objekt har i exakt samma medlemmar och händelsehanterare som `text`-objektet. Det enda som skiljer dem åt är att all text blir till asterisker, då en besökare skriver in något i en `password`-ruta. Det är väl tänkt så att ingen skall kunna se över någons axel vad man har för lösen på någon personlig sida som till exempel hotmail.com. Behandla denna på samma sätt som ett `text`-objekt!

Objektet 'textarea'

Detta objekt kan du se som en utökad textruta med flera rader och kolumner. Exakt samma medlemmar och händelsehanterare ingår!

Objektet 'hidden'

Objektet har inga medlemsfunktioner eller händelsehanterare, endast datamedlemmarna är desamma som för `text`-objektet. Man använder sig av `hidden`-objektet som en slags möjlighet att skicka "osynlig" information med ett formulär.

Knapp-objekt i formulär

Objekten 'button', 'reset', 'submit'

Det finns tre "olika" typer av knappar i formulär, vilka ändå är väldigt lika varandra. Dessa tre är knapparna `button`, `reset` och `submit`. När du ser en rektangulär, grå knapp i ett formulär är denna av någon av dessa nämnda typer. Det enda som skiljer dem åt är att de avfyrar olika händelser då de trycks ned. Knapparna `reset` och `submit` fungerar så att *onReset* respektive *onSubmit* avfyras vid knapptryck, därefter sköter webbläsaren själv det som knappen skall utföra – dvs antingen återställa formuläret eller skicka iväg dess innehåll. Knappen `button` har förstås inte dessa händelsehanterare, däremot har den *onClick* som man kan använda mycket effektivt. Förutom att händelsehanterarna för de tre knapparna skiljer sig åt är de identiska.

`button`-, `reset`- och `submit`-objekten:

Datamedlemmar	Medlemsfunktioner	Event-handlers
`name`	`click()`	`onClick=`
`type`		`onMouseDown=`
`value`		`onMouseUp=`

Datamedlemmar

name

Denna medlem motsvarar det namn du ger en knapp i `<input>`-taggen då du skapar formuläret. Man kan faktiskt ändra namnet via ett skript, även om jag har svårt att tro att det någonsin blir aktuellt. Däremot kan det vara av värde att kunna ta fram namnet på den knapp som trycktes ned i ett formulär (skulle kunna användas till enklare spelfunktioner). Titta på koden nedan. Här visar jag hur knapparna i formuläret namnges och hur de sedan anropar `whatButton()` då någon trycker på dem. Observera att jag använder det reserverade ordet `this` som parameter i funktionsanropet, detta refererar till den aktuella knappen.

```
<INPUT TYPE="button" name="knapp1" value="knapp1"
onClick="whatButton(this)">

function whatButton(clickedButton)
{
   var theButton = clickedButton.name;
   alert("Du klickade på: " + theButton);
}
```

type
Som ovan, men man kan inte ändra på typen via ett skript. Kontrollera typen på ett formulärobjekt med något liknande

```
theType = theForm.elements[0].type;
```

value
Detta är det som står på knappen – exempelvis ”Skicka”. Du kan ändra detta via skript, men jag tror inte man har någon särskild användning av det. Om man har någon användning av denna medlem så bör det vara på liknande vis som för `name`.

Medlemsfunktioner

click()
Funktionen skall simulera exakt detsamma som då en användare trycker på den aktuella knappen. Skriv alltså något liknande

```
document.forms[0].knapp3.click();
```

Event-handlers

onClick har vi ju använt (även till andra objekt), men det finns två andra som vi ännu inte stiftat närmare bekantskap med – dessa är *onMouseDown* och *onMouseUp*.

onMouseDown, onMouseUp
Dessa är aningen mer väldefinierade än *onClick*, i det avseendet att sammansättningen av ett *onMouseDown* och ett *onMouseUp* ger ett *onClick*. Prova själv att klicka på en knapp och rulla pekaren från knappen medan du håller musknappen nere. Knappen kommer att återgå från nedtryckt läge till sitt normala läge – utan att *onClick* avfyras!

Objektet ’checkbox’

Detta objekt är en ”bockruta” som man ofta använder på sidor där man exempelvis får fylla i sina intresseområden (då man ansöker om medlemskap på en portal eller liknande). Man kan skapa en hel rad checkboxar, vilka ingen, några eller alla kan vara förbockade samtidigt. Förutom de medlemmar som de tidigare nämnda knapparna har, så har detta objekt även medlemmarna `checked` och `defaultChecked`.

checked, defaultChecked
Då du klickar i en checkbox kommer dess medlem `checked` att få värdet `true` – den har annars värdet `false`. Du kan då med ett skript undersöka om en särskild ruta är förbockad via ex.

```
if (document.forms[0].box_1.checked) // satser om förbockad
```

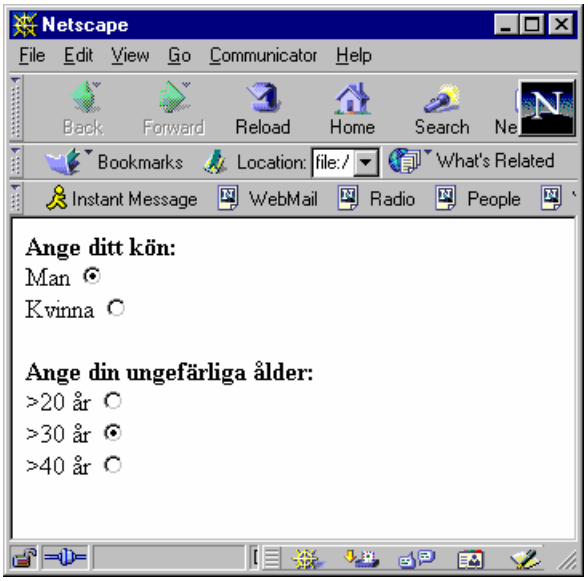
Du kan också själv bocka för en ruta via ett skript med nåt liknande

```
document.forms[0].box_1.checked = true;
```

När du skapar en checkbox kan du välja att den skall vara förbockad från början genom att lägga till ordet `checked` i `<input>`-taggen. Om du gör det kommer medlemmen `defaultChecked` att få värdet `true`. Denna går däremot inte att ändra på via skript (förstås).

Objektet 'radio'

Denna knapp fungerar lite annorlunda än checkboxen. Den är tänkt att efterlikna gammaldags radioknappar, där man valde *en* av flera funktioner. Dvs om det fanns tre knappar att välja på kunde man bara välja en åt gången, och en eventuellt vald knapp trycktes upp om man tryckte på en annan. Knappen har samma medlemmar som checkboxen och dessutom en till – `length`. Denna innehåller ett heltal som motsvarar det antal radioknappar som ingår i gruppen. Då man skapar en uppsättning radioknappar, som skall få den ovan nämnda egenskapen, måste man namnge *alla* knapparna med samma namn. Betrakta formuläret nedan:



Man skapar det genom

```
<FORM>
<B>Ange ditt kön:</B><BR>
Man <INPUT TYPE="radio" name="sex"><BR>
Kvinna <INPUT TYPE="radio" name="sex"><BR><BR>
<B>Ange din ungefärliga ålder:</B><BR>
>20 år <INPUT TYPE="radio" name="age"><BR>
>30 år <INPUT TYPE="radio" name="age"><BR>
>40 år <INPUT TYPE="radio" name="age"><BR>
</FORM>
```

Lägg märke till att jag namnger de övre två knapparna med "sex" och de undre tre med "age". Detta talar om för webbläsaren vilka knappar som hör ihop, och hur en besökare skall kunna välja med dem. Det är uppenbart att man antingen är man eller kvinna och inte både och. Här passar radioknappen utmärkt. Den passar lika bra till att bestämma ungefärlig ålder också –

besökaren uppmanas indirekt att göra en så noggrann uppskattning som möjligt av sin ålder. Att undersöka hur många radioknappar som ingår i `age`-gruppen är enkelt genom

```
var lengd = document.forms[0].age.length;
```

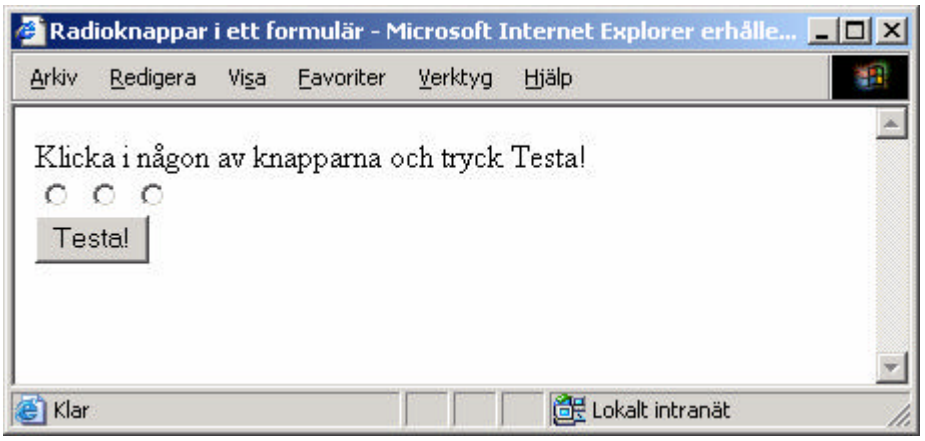
Om du vill åt `name`-attributet för en radioknapp måste du referera till en enskild knapp (som alltid annars) – inte till gruppen.

```
var namnet = document.forms[0].elements[0].name;
```

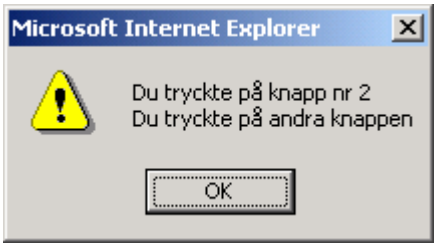
Kom också ihåg att du kan kontrollera vilken radioknapp som är nedtryckt genom att undersöka om egenskapen `checked` är `true`

```
if (document.forms[0].age[0].checked == true) // satser om klickad
```

I exemplet nedan använder jag tre radioknappar för att demonstrera hur man söker genom en uppsättning med okänt antal radioknappar. Idén är att man skall klicka i en av radioknapparna och därefter klicka på knappen ”Testa!”. Man får då information – på två olika sätt – i en alert-box om vilken knapp man tryckte på. Så här ser fönstret ut när man först laddar det:



När man sedan väljer en knapp och testar får man ett meddelande:



Man kan tycka att det övertydligt att meddela samma information på två sätt, men det gör jag förstås bara för att demonstrera tekniken. Nedan listar jag den centrala delen av koden till sidan:

```
<form name="form1" method="post" action="">
  Klicka i n&aring;gon av knapparna och tryck Testa!<br>
<input type="radio" name="radiobutton" value="första knappen">
<input type="radio" name="radiobutton" value="andra knappen">
<input type="radio" name="radiobutton" value="tredje knappen"><br>
<input type="button" name="" value="Testa!" onClick="knappTest()">
</form>
```

```
function knappTest()
{
  var i = 0;
  while ((document.form1.radiobutton[i].checked != true) &&
           (i < document.form1.radiobutton.length))
    i++;
  alert ("Du tryckte på knapp nr " + (i + 1) + "\nDu tryckte på "
    + document.form1.radiobutton[i].value);
}
```

Skriptet ligger som vanligt i `<HEAD>` sektionen. Som du ser har alla radioknappar fått samma namn. Glöm inte detta – det är en förutsättning för att de ska fungera. Till varje knapp har jag också lagt till ett `value`-attribut som avslöjar vilken knapp i ordningen det handlar om. Dessa kan jag sedan komma åt med ett JavaScript. Så till funktionen `knappTest()`. Där börjar jag med att initiera räknevariabeln i till värdet 0. Denna används för att loopa igenom knapparna i tur och ordning. `while`-satsens innebörd är ”*så länge* radioknappen med index `i` *inte* är klickad *och* `i` är *mindre* än antalet knappar i gruppen, så …”. Den enda satsen som körs om villkoren är uppfyllda är `i++;` vilken endast ökar i med ett steg. Det är ju naturligt, eftersom om en knapp inte är klickad är det ju dags att undersöka om *nästa* knapp är klickad. Denna procedur fortgår fram till (men inte med) då `i` antar värdet 3. Detta eftersom det finns tre radioknappar med namnet `radiobutton` och `document.form1.radiobutton.`**`length`** får då värdet 3.

Information om vilken knapp som är klickad kan nu ges på ett av två sätt – antingen kan man skriva

```
alert ("Du tryckte på knapp nr " + (i + 1));
```

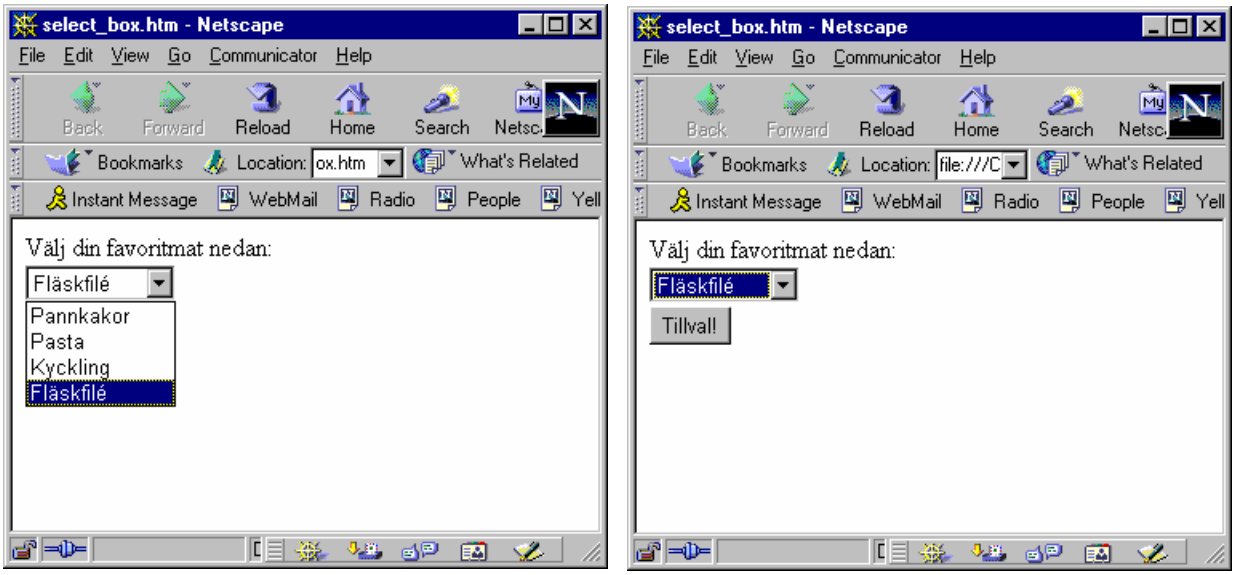
eller så skriver man

```
alert(”Du tryckte på "  + document.form1.radiobutton[i].value);
```

I det första alternativet måste man lägga till 1 till `i`, eftersom index i alla vektorer i JavaScript går från 0 och uppåt (den tredje knappen har alltså index 2). I det andra alternativet utnyttjar jag `value`-attributet till varje knapp. Jag kommer åt denna sträng med uttrycket `document.form1.radiobutton[i].value`. Märk väl att jag här använder `i` som den är, utan att lägga till 1.

Objektet ’select’

En `select`-box är en ganska komplicerad formulärkomponent. Du har säkert sett rullgardinsmenyer på webbsidor där man kan välja ett av flera val genom att klicka i menyn. Ungefär som nedan



Genom att använda dessa rullgardiner kan man tjäna en del utrymme på en sida. Det finns även de som använder `select`-boxar som navigeringsverktyg på sajter (mer om det längre fram). Låt oss först ta en titt på några av objektets medlemmar.

`select`-objektet:

Datamedlemmar	Medlemsfunktioner	Event-handlers
`length`	`blur()`	`onChange=`
`name`	`focus()`	`onFocus=`
`selectedIndex`		`onBlur=`
`options[i].text`		
`options[i].value`		

Datamedlemmar

length, name

Medlemmen `name` refererar förstås till den sträng du angav i `name`-attributet då du skapade `select`-boxen i HTML-koden för sidan. Medlemmen `length` är ett heltal som motsvarar det antal element som finns i `select`-boxens lista. I exemplet ovan har `length` värdet 4.

selectedIndex

Så fort att en besökare klickar i en rullgardinslista kommer medlemmen `selectedIndex` att sättas till det heltal som valet motsvarar. Det första listelementet har index 0 som vanligt.

options[i].text, options[i].value

När du skapar en rullgardinslista har du möljighet att definiera både den text (`text`) som skall visas i listan, och ett särskilt värde (`value`) som förblir dolt men som är tillgängligt via skript. Betrakta koden nedan

```
<FORM>
Välj din favoritmat nedan:<BR>
<select name="lista" size="1">
  <option value="Pannkakor med sylt!" selected>Pannkakor
  <option value="Spaghetti med köttfärssås!">Pasta
  <option value="Kyckling med ris!">Kyckling
```

```
    <option value="Fläskfilé med potatisgratäng!">Fläskfilé
</select><BR>
<INPUT TYPE="button" onClick="goSelected(this.form)" value="Tillval!">
</FORM>

function goSelected(form)
{
   alert(form.lista.options[form.lista.selectedIndex].value);
}
```

Jag definierar hela listan med `<select>`-taggen. Därefter definierar jag varje listelement med `<option>`-taggen och till denna finns attributen `value` och `text`. Det som hamnar i `text`-attributet kommer att synas i listan. Sedan har jag lagt till en knapp (`button`) för att demonstrera hur man kommer åt dessa två attribut via skript. (Funktionen `goSelected()` kan man lägga i `<head>`-sektionen). Vi försöker nu att bena ut den något knepiga referensen i funktionen ovan. Som parameter i funktionsanropet anger jag `this.form`, dvs *hela* det formulär som listan ingår i. I funktionshuvudet kommer denna referens att tilldelas namnet `form` – det betyder att `form.lista` refererar till den aktuella rullgardinsmenyn. Det betyder också att `form.lista.selectedIndex` är det heltal som representerar det markerade valet i listan (första elementet har värdet 0). Dvs den senare delen av hela uttrycket blir `options[i].value`, där `options[i]` refererar till det aktuella listelementet, och där `value` är den sträng som man skrivit in i `value`-attributet då man skapade formuläret. Krångligt? Det lossnar snart! Så om någon väljer fläskfilé och trycker på knappen ”Tillval!”, så kommer `alert`-boxen att visa strängen ”Fläskfilé med potatisgratäng!”.

Om jag istället byter ut `value` mot `text` i referensen kommer förstås samma sträng att visas i `alert`-boxen, som den sträng listan visar. Medlemsfunktioner och event-handlers fungerar precis som de gör för de objekt som jag visat tidigare. Med alla dessa formulärkomponenter är vi snart redo att göra riktigt kul saker på våra sidor.

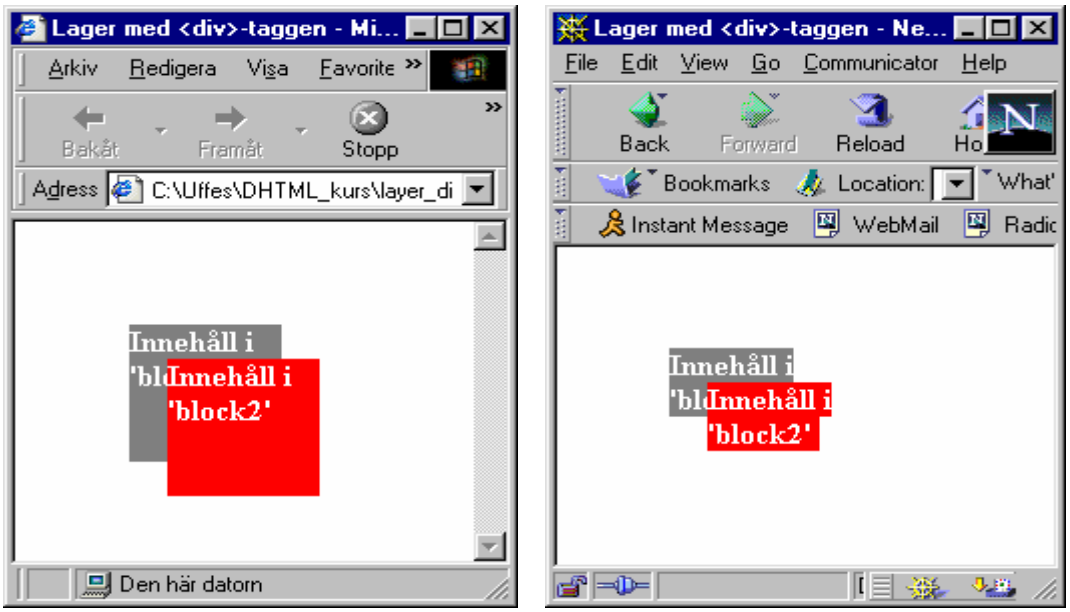
Lager med div-taggen

Om du har sysslat med HTML ett tag kommer du att märka de stora fördelarna med att använda block (eller *divisions*) för att presentera innehåll i en sida. Detta eftersom det finns oerhörd flexibilitet över *hur* innehållet i ett block skall presenteras – det är nämligen du själv som bestämmer! Du kan exempelvis bestämma exakt placering av ett block, om ett block skall vara synligt eller inte, du kan lägga block ovanpå varandra, omlott med varandra etc. Det är dessutom möjligt att ge blocket en bakgrundsfärg eller bakgrundsbild osv. Det är i detta sammanhanget jag väljer att kalla sådana objekt *lager*.

Med lager i webbsidor har designen tagit relativt stora steg från den statiska representation av text och bild som HTML tillåter. Du kan tänka dig ett lager ungefär som ett genomskinligt OH-papper som du kan placera var som helst på sidan. Ett lager innehåller väsentligen HTML-objekt som bilder, text och formulär eller eventuellt ytterligare (nästlade) lager. Använder du endast lager

och positionering via CSS för statisk presentation får du säkert inte så mycket bekymmer. Det är när du vill styra objekten under skriptkontroll som problemen uppstår. Dels därför att det är lite klurigt och dels för att de stora aktörerna har valt att låta sina webbläsare implementera lager på väldigt olika sätt. Om du känner till NN vet du kanske att det fr. o m. NN4 fanns ett märke som hette just `layer`. Denna motsvarighet finns inte ens i IE4 och det är nog lika bra det! Det var fruktansvärt omständigt, om än logisk korrekt, att referera mellan lager med NN:s gamla lagerimplementering. Av denna anledning kan man alltså inte referera till ett lager och dess komponenter och egenskaper på samma sätt i NN4 som i IE4 (och framåt). Dessutom är det så att lager fungerar bättre i IE än i NN av någon okänd anledning. Tanken är att NN6 (vart tog NN5 vägen?) skall fungera enligt samma modell som IE.

Eftersom `layer`-taggen är NN specifik och dessutom i dessa dagar helt onödig tas den inte upp här. Det verkar som att NN endast har några enstaka procent (ca 3 %) av Internetpubliken (5/8 - 03), alltså kan man med hyfsat samvete tillverka sidor uteslutande för IE. Vi kommer att skapa lager mha `<div>`-taggen, vilken både NN och IE hanterar, men skillnader i hur IE och NN renderar sådana lager är ibland fortfarande klart irriterande. Man använder `<div>`-taggen tillsammans med stilmallar som antingen definieras inom `<style>`-taggen i `<head>`-sektionen, eller med `style`-attributet direkt i `<div>`-taggen. Tag följande enkla exempel:




```
<HTML>
<HEAD>
<TITLE>Lager med &lt;div&gt;-taggen</TITLE>
<style type="text/css"><!--
#block1 {
          position: absolute;
          top: 0.5cm;
          left: 0.5cm;
          width: 2cm;
          height: 2cm;
          background-color: gray;
          visibility: visible;
          }
#block2 {
          position: absolute;
          top: 1cm;
```

Ulf Tornert
ulftt@gy.varmdo.se

```
            left: 1cm;
            width: 2cm;
            height: 2cm;
            background-color: red;
            visibility: visible;
            }
// --></style>
</HEAD>
<BODY>
<div id="block1">
<FONT COLOR="#FFFFFF"><B> Innehåll i 'block1'</B></FONT>
</div>
<div id="block2">
<FONT COLOR="#FFFFFF"><B> Innehåll i 'block2'</B></FONT>
</div>
</BODY>
</HTML>
```

I IE har vi inget att anmärka på, men i NN ser det ut som ett världskrig! Att anpassa dokument för de olika webbläsarna brukar kallas ”cross-browser-scripting” och kan vara något av det mest irriterande som finns.

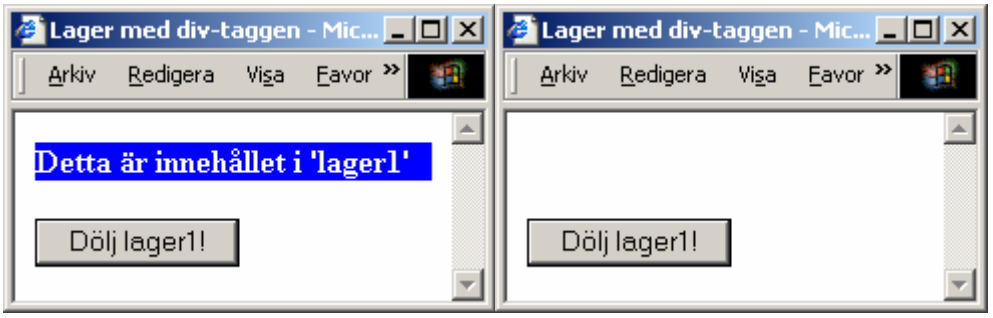
Det som är intressant för oss just nu är att skapa lager som går att manipulera via ett JavaScript. Av någon okänd anledning har NN ibland uppenbara problem att hantera lager som skapats med inline-mallar, dvs säga så här

```
<div id=”lager1” style=”background-color: yellow”>Lager1</div> // Nej!
```

Det är därför en mycket god idé att skapa reglerna till lagren uppe i sidhuvudet istället. För att belysa tekniken med hur man refererar till lager och dess egenskaper kan du studera koden nedan

```
<HTML>
<HEAD>
<TITLE>Lager med div-taggen</TITLE>
<style type="text/css">
#lager1 {
   background-color: blue;
   color: white;
}
</style>
<SCRIPT LANGUAGE="JavaScript">
function notShowing()
{
   document.getElementById("lager1").style.visibility = "hidden";
}
</SCRIPT>
</HEAD>
<BODY>
<div id="lager1"><b>Detta är innehållet i 'lager1'</b></div>
<P><FORM>
<INPUT TYPE="button" value="Dölj lager1!" onClick="notShowing()">
</FORM></P>
</BODY>
</HTML>
```

Sidan ser ut som nedan. Till vänster innan man tryckt på knappen och till höger efter.



Lagret blir så brett som sidan är och det kommer att utökas vertikalt så att innehållet i sin helhet visas. Det finns ett sätt att förhindra denna utökning genom att samtidigt sätta `overflow`-attributet till något av värdena `auto`, `scroll` eller `hidden`. Värdet `scroll` ger som namnet antyder scrollknappar till lagret så att resten av innehållet kan läsas av (värdet `auto` ger som regel endast en vertikal scrollknapp). Om du sätter värdet till `hidden` måste du veta att det inte finns något sätt att se det som hamnar utanför lagrets dimensioner – var försiktig!

I formuläret ger jag besökaren möjlighet att ändra på synligheten för lagret genom att anropa funktionen `notShowing()` via knappens *onClick*. I den funktionen använder jag nu metoden `document.getElementById()` för att skapa en referens till lagret.

```
document.getElementById("lager1").style.visibility = "hidden";
```

Detta skrivsätt är relativt komprimerat och det går att skriva isär det så här

```
var lagret = document.getElementById("lager1");
lagret.style.visibility = ”hidden”;
```

Variabeln `lagret` är alltså en referens till själva `div`-objektet och vi använder sedan detta alias för att ändra på ett `style`-attribut (`visibility`). Lägg även märke till att metoden `getElementById()` hör till objektet `document`. Anta att du istället vill ändra på bakgrundsfärgen på lagret. Då skriver du

```
lagret.style.backgroundColor = "red";
```

Obs! Notera att jag måste använda `backgroundColor` för att ändra färgen och inte `bgColor` som man skulle kunna tro. Mycket förvirrande! Objekt som skapas med `div`-taggen kan ändra sina CSS-egenskaper via skript *om* man känner till vad motsvarande egenskap heter i JavaScript. När lagret ovan först definieras görs detta med

```
#lager1 {
   background-color: blue;
   color: white;
}
```

men när jag sedan ändrar bakgrundsfärgen med ett skript görs detta med

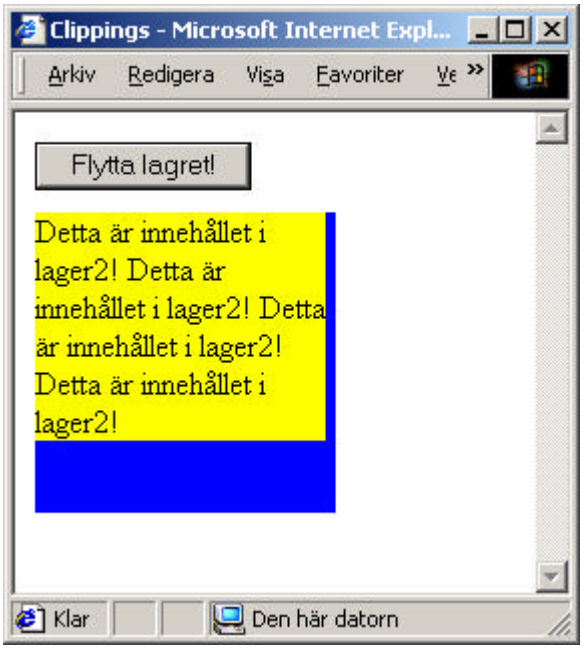
```
lagret.style.backgroundColor = "red";
```

Man brukar säga att `backgroundColor` är *JavaScript-ekvivalenten* till `style`-attributet `background-color`. Här kan det vara på sin plats med ett upplagsverk och som tur är finns

detta i Dreamweaver! När du har DW öppet väljer du Fönster > Referens. Där hittar du troligtvis alla JavaScript-ekvivalenter du någonsin kommer att behöva. Om inte – sök på webben!

Att sätta clip-regionen till lager

Style-attributet `clip` har också att göra med den synliga arean i ett lager. Som jag ser det har man som mest nytta av clippings när man nästlar lager, dvs man har ett lager i ett annat lager. För att demonstrera `clip`-attributet skapar jag följande sida

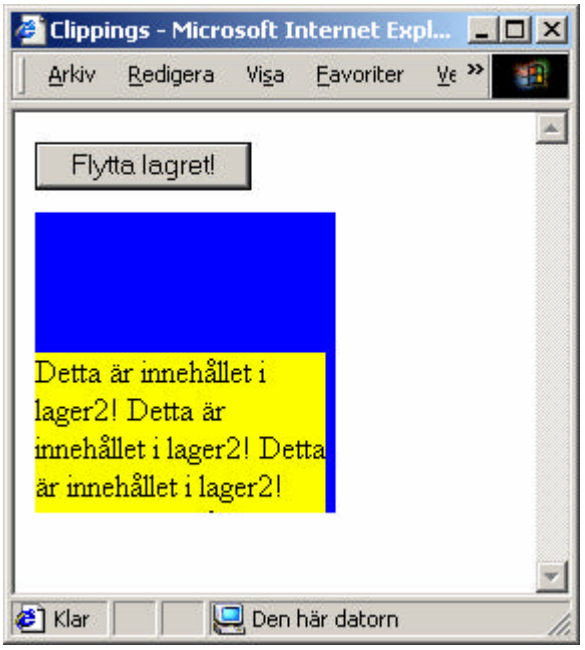


med koden

```
<HTML>
<HEAD>
<TITLE>Clippings</TITLE>
<style type="text/css">
#lager1 {
   position:absolute;
   top: 50px;
   background-color: blue;
   width: 150px;
   height: 150px;
   clip: rect(0px 150px 150px 0px);
}

#lager2 {
   position: absolute;
   background-color: yellow;
}
</style>
<SCRIPT LANGUAGE="JavaScript"><!--
function move()
{
   var lagret = document.getElementById("lager2");
   lagret.style.pixelTop += 10;
}
```

```
// --></SCRIPT>
</HEAD>
<BODY>
<div id="lager1">
<div id="lager2"><!-- Ett nästlat lager -->
Detta är innehållet i lager2! Detta är innehållet i lager2!
Detta är innehållet i lager2! Detta är innehållet i lager2!
</div>
</div>
<P><FORM>
<INPUT TYPE="button" value="Flytta lagret!" onClick="move()">
</FORM></P>
</BODY>
</HTML>
```

Koden åstadkommer följande resultat



Effekten blir att det innersta lagret kommer att beskäras vid den nedre kanten av det yttre, omgivande lagret. Om vi inte satt clippings skulle det innersta lagret helt enkelt flyttas ut ur det omgivande lagret och det var ju inte tanken. Nu kanske någon säger: ”Varför inte använda attributet `overflow` istället?”. Det är en bra invändning eftersom effekten i det här exemplet skulle blivit identisk, men konstruktionen `clip: rect(top right bottom left)` har större möjligheter att bestämma var någonstans beskärningen skall äga rum. De fyra värdena som anges går medsols med början på övre kanten och anger vid vilken pixel (i lagrets koordinatsystem) innehållet skall beskäras. Det betyder att

```
clip: rect(0px 150px 150px 0px);
```

beskär vid 0:te pixeln vid övre kanten, vid den 150:e pixeln i höger kant etc.

Ulf Tornert
ulftt@gy.varmdo.se

Event-handlers till lager

När en besökare på en sida rullar över `clip`-regionen för något lager med muspekaren kommer händelsen *onMouseOver* att avfyras. På samma sätt kommer *onMouseOut* att avfyras då muspekaren lämnar lagrets `clip`-region. Om jag skriver

```
<div id="lager1" onMouseOver=”alert(’Musen över lager1’)”>
```

kommer jag att få en liten hälsning så fort jag rullar in muspekaren över `lager1`.

Objektet ’Math’

Även om de flesta aldrig kommer att utnyttja kraften i detta objekt så kan det vara idé att känna till något om det. Detta objekt innehåller allt sådant som att generera slumptal, avrunda decimaltal, bestämma det största av två tal, kvadratrotsberäkningar osv. Objektet har bara datamedlemmar och medlemsfunktioner, varav *några* är

`Math`-objektet:

Datamedlemmar	Beskrivning
`Math.E`	`Eulers konstant (?2,7182818...)`
`Math.PI`	`? (?3,1415926...)`
`Math.SQRT2`	$\sqrt{2}$ `(?1,4142135...)`

Medlemsfunktioner	Returnerar
`Math.abs(x)`	`Storleken på x utan tecken`
`Math.ceil(x)`	`Nästa heltal >= x`
`Math.floor(x)`	`Nästa heltal <= x`
`Math.max(x, y)`	`Det största talet av x och y`
`Math.min(x)`	`Det minsta talet av x och y`
`Math.random()`	`Slumptal mellan 0 och 1`
`Math.round(x)`	`Avrundat heltal från x`
`Math.sqrt(x)`	`Kvadratroten ur x`

Dessutom finns ju alla de trigonometriska funktionerna `Math.sin(x), Math.cos(x), Math.tan(x)` och deras inverser `Math.asin(x), Math.acos(x), Math.atan(x)` och så vidare. Samtliga funktioner använder vinkelmåttet radianer. En ”vanlig” uppgift är att ta fram ett slumptal mellan ett givet intervall. Antag att du skall slumpa fram ett tal mellan 1 och 4 {1, 2, 3, 4} så kan detta göras med satsen

```
var tal = Math.round(Math.random()*4);
```

`Math.random()` tar fram ett värde mellan 0 och 1, sedan multiplicerar vi detta tal med 4. Då får vi som minst 0 och som mest 4, därefter avrundar jag med `Math.round()` till ett heltal i detta intervall. Om jag vill skapa ett slumptal mellan 1 och 5 så kan jag ju bara lägga på 1 till det talet jag fick:

```
tal += 1;
```

Dessa två satser kan sammanföras till ett enda uttryck

```
var tal = Math.round(Math.random()*4) + 1;
```

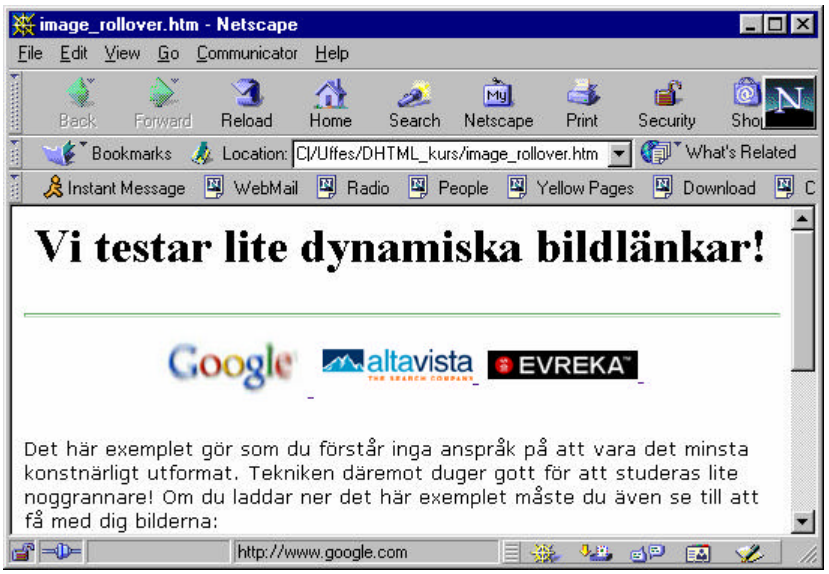
Jag hänvisar till mer utförlig litteratur för den som är intresserad av att veta mer om `Math`-objktet.

Några enkla exempel

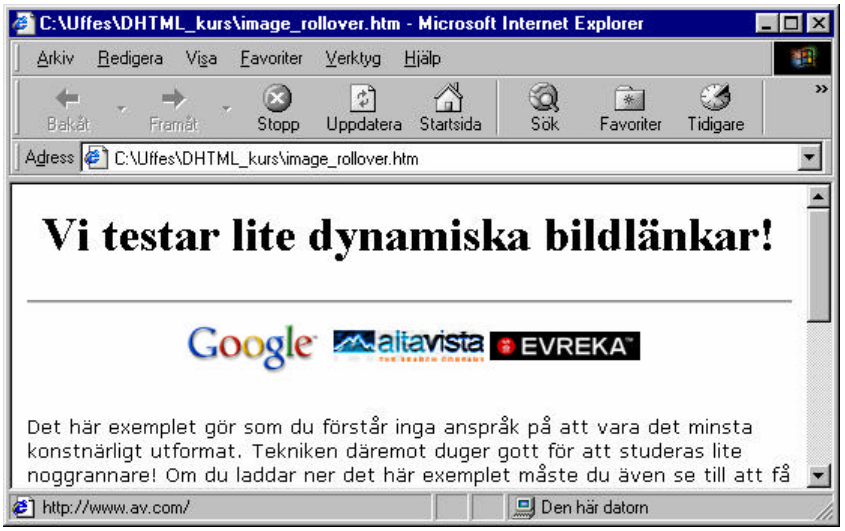
I den här sektionen kommer jag att ta upp några enkla, ofta förekommande skript för lite varierande situationer. De är tänkta att ge dig en lite större bild av hur man kan använda JavaScript för att åstadkomma mer med en webbsida än vad man kan med bara standard HTML. Jag kommer inte att kommentera koden *särskilt* utförligt, utan lägga tonvikten där jag tror att en förklaring kan vara på sin plats. För att du skall få så mycket ut av den här delen som möjligt har jag lagt ut samtliga av dessa skript på adressen **`http://hem.passagen.se/ulftt`**. Genom att ladda ner skripten och manipulera med koden kan du lära dig ännu mer. I nästa sektion kommer du att få träna dig på att göra dina egna skript, då kan du säkert få lite nytta av att först ha studerat den här delen.

Image rollover

I de allra flesta webbsidor kunde man, åtminstone för ett tag sedan, erfara hur bilder förändrade sitt utseende då man rullade över dem med muspekaren. Det här exemplet visar hur man kan göra det. Det kan vara en god idé att tänka igenom hur man skall lägga upp sidan, innan man skapar och namnger de bildfiler som skall ingå i den. Ofta kan man skapa lättförståelig logik om man bara ”skyndar långsamt” och ”tänker efter före”. När man skapar den här typen av effekter är det viktigt att man *förladdar* bilderna innan man exekverar de skript som förändrar utseendet i sidan. Det gör man eftersom en bild inte läggs till ”cachen” förrän den lästs in i browsern. Om en bild läses in samtidigt som man vill ha en ögonblicklig förändring kommer man att uppleva en irriterande fördröjning (om bilderna är tillräckligt stora). Vad man sedan väsentligen gör är att man byter ut en bild mot en annan, genom att manipulera med bildens källfil via `image.src`. För att kunna ändra en bild vid den händelsen att muspekaren står över den, måste man ju fånga den händelsen. `Image`-objektet genererar inte en sådan händelse och har heller ingen sådan event-handler, så vi använder en *länkens onMouseOver* istället (rollover-bilder är ju oftast länkar). Jag visar först hur sidan ser ut och därefter kommer koden. I NN får vi följande resultat:



och i IE ser det ut så här



I exemplen ovan har jag först rullat över länken till Google och sedan länken till Altavista. Koden som genererar sidan kommer nedan.

```
<HTML>
<HEAD>
<TITLE>image_rollover.htm</TITLE>
<SCRIPT LANGUAGE="JavaScript"><!--
// Skapar vektorer för att kunna förladda bilderna och
// göra automatiserade anrop vid bildbyten.
// Vektorn imageOn innehåller alla de bilder som skall
// länkas då pekaren står över bildlänkarna, och imageOff
// innehåller de bilder som är länkade (från början).

var imageOn = new Array(3);
for (i = 0; i < 3; i++)
   imageOn[i] = new Image();

var imageOff = new Array(3);
for (i = 0; i < 3; i++)
```

```
    imageOff[i] = new Image();

// Förladdar bilderna så att det går fortare att byta sedan.
// Om du laddar ner filen för att studera skriptet i lugn och ro
// måste du även få med dig alla dessa sex bilder.

imageOn[0].src = "google_on.jpg";
imageOn[1].src = "av_on.jpg";
imageOn[2].src = "ev_on.gif";
imageOff[0].src = "google_off.gif";
imageOff[1].src = "av_off.gif";
imageOff[2].src = "ev_off.gif";

// Detta är den funktion som byter bild vid onMouseOver
function imageTurnOn(nr)
{
  document.images[nr].src = imageOn[nr].src;
}

// Detta är den funktion som byter bild vid onMouseOut
function imageTurnOff(nr)
{
  document.images[nr].src = imageOff[nr].src;
}
// --></SCRIPT>
</HEAD>
<BODY><center>
<H1>Vi testar lite dynamiska bildlänkar!</H1>
<HR>
<table>
<tr>

  <td><a href="http://www.google.com" onMouseOver="imageTurnOn(0)"
  onMouseOut="imageTurnOff(0)"><img border="0" src="google_off.gif">
  </a></td>

  <td><a href="http://www.av.com" onMouseOver="imageTurnOn(1)"
  onMouseOut="imageTurnOff(1)"><img border="0" src="av_off.gif">
  </a></td>

  <td><a href="http://evreka.passagen.se" onMouseOver="imageTurnOn(2)"
  onMouseOut="imageTurnOff(2)"><img border="0" src="ev_off.gif">
  </a></td>

</tr>
</table>
</center>
<P><FONT SIZE="2" FACE="verdana, helvetica, sans-serif">Det här exemplet
gör som du förstår inga anspråk på att vara det minsta konstnärligt
utformat. Tekniken däremot duger gott för att studeras lite
noggrannare!</FONT></P>
</BODY>
</HTML>
```

Som du ser är funktionerna som byter bilderna förvånansvärt enkla – det beror på att jag har strukturerat bildvektorerna `imageOn` och `imageOff` så att dess index motsvarar `image[]`-vektorns index i `document`-objektet. När jag anropar funktionen `imageTurnOn(nr)` så vet jag ju vilket index den bild jag vill förändra har. Om jag ser till att detta index motsvarar rätt bild i vektorn `imageOn` kommer satsen

Ulf Tornert
ulftt@gy.varmdo.se

```
document.images[nr].src = imageOn[nr].src;
```

att sköta allting automatiskt. När jag från den första bildlänken (med index 0) anropar `imageTurnOn(nr)`, så vet jag ju att det är `document.images[0].src` jag vill förändra – alltså sätter jag `nr = 0`. Satsen ovan blir då

```
document.images[0].src = imageOn[0].src;
```

Dvs den första bilden i dokumentet kommer att tilldelas den första bilden i vektorn imageOn. Där ligger bilden ”`google_on.jpg`” – precis den jag vill ha! På samma sätt gör jag för de andra två bildlänkarna, vilka har index 1 respektive 2.

Formulärkontroll

Det här området kan man göra hur stort som helst, så jag kommer bara att presentera ett kortare exempel! Tanken med att kontrollera ett formulär är i alla fall att besökaren skall ha skrivit i meningsfulla data i ett formulär *innan* det skickas. Om ett formulär innehåller en textruta där man skall fylla i sitt förnamn, så är värdet ”Ul,f” inte önskvärt. Att värdena är korrekta innan formuläret skickas har ännu större betydelse om de ska stoppas in i en databas på serversidan där formulärdatat tas emot. En stor fördel med att kontrollera formulär med JavaScript är ju att alla kontroller sker på besökarens dator – vilket inte belastar nätet alls! Det finns ju förstås möjligheter att kontrollera formulären på serversidan också, men då måste formulärdatat skickas fram och tillbaka tills allt är korrekt ifyllt och det är onödigt och oekonomiskt. Dessutom måste du programmera ett CGI-skript (om du inte hittar ett gratis som löser det åt dig) och hitta en ”webhost” som låter dig använda det skriptet. Nackdelen med JavaScript är att det går att komma runt en kontroll med lite kunskap, och då faller ju hela idén. Så slutsatsen blir att man bör göra lättare formulärkontroll med JavaScript innan formuläret skickas, därefter kan man göra en djupare kontroll (om det behövs) på serversidan. Om något saknas eller är felaktigt kan man be besökaren ändra på detta och skicka iväg formuläret igen. I exempelt nedan simulerar jag en sida där man kan poängsätta en hel sajt med ett tal från 1 till 10. Man *måste* då fylla i sitt namn och e-post, och betyget *måste* vara ett tal mellan 1 och 10. Ett sådant formulär skulle kunna se ut så här:

Ulf Tornert
ulftt@gy.varmdo.se

I exemplet har jag skrivit in namnet och e-post adressen korrekt, men gett sajten ett otillåtet betyg. Koden som genererar denna sida är

```
<HTML>
<HEAD>
<TITLE>formcheck.htm</TITLE>
<SCRIPT LANGUAGE="JavaScript"><!--
```

Ulf Tornert
ulftt@gy.varmdo.se

```
function checkForm()
{
  var form = document.forms[0];
  if (form.theName.value == "" && form.email.value == ""
        && form.grade.value == "")
             return false; // skicka inte tomma formulär
  else if (form.theName.value == "" || form.theName.value == null)
  {
    alert('Du måste fylla i ditt namn');
    form.theName.focus();
    return false;
  }
  else if (form.email.value == "" || form.email.value == null)
  {
    alert('Du måste fylla i din\ne-post adress');
    form.email.focus();
    return false;
  }
  else if (form.grade.value == "" || form.grade.value == null)
  {
    alert('Sätt betyg på vår sajt :)');
    form.grade.focus();
    form.grade.select();
    return false;
  }
  else if (form.grade.value.length > 2 || parseInt(form.grade.value) < 1
             || parseInt(form.grade.value) > 10
             || parseInt(form.grade.value).toString() == "NaN")
  {
    alert('Fyll i ett betyg mellan 1 och 10!');
    form.grade.focus();
    form.grade.select();
    return false;
  }
  else return true;
}
// --></SCRIPT>
</HEAD>
<BODY>

<H1>Sätt betyg på vår sajt!</H1>
<HR>
<FORM onSubmit="return checkForm()">
Fyll i ditt namn: 
<INPUT TYPE="text" name="theName"><BR>
Fyll i din e-post adress 
<INPUT TYPE="text" name="email"><BR>
Vilket betyg ger du vår sajt (från 1 till 10)? 
<INPUT TYPE="text" name="grade" size="2"><BR>
<INPUT TYPE="reset" name="reset" value="Återställ">
<INPUT TYPE="submit" name="submit" value="Skicka">
</FORM>

</BODY>
</HTML>
```

När man använder formulärets *onSubmit*-hanterare är det viktigt att komma ihåg nyckelordet `return`. Hanteraren vill nämligen veta om den skall skicka formuläret eller inte efter det att

funktionen `checkForm()` är färdig. Formuläret kommer att skickas om funktionen returnerar `true`, men om den returnerar `false` händer ingenting! Det är därför jag i samtliga tester returnerar `false` med satsen `return false`. Slutligen om alla testvillkoren, ett efter ett, *inte* är uppfyllda kommer funktionen att returnera `true`. Lägg märke till att en textruta är korrekt ifylld om motsvarande testvillkor *inte* är uppfyllt. Dvs textrutan är då *inte* tom. För betyget har jag skapat en mer heltäckande test – denna kan behöva förklaras mer i detalj. Testen har följande utseende

```
if (form.grade.value.length > 2 || parseInt(form.grade.value) < 1
              || parseInt(form.grade.value) > 10
              || parseInt(form.grade.value).toString() == "NaN")
```

och de respektive delarna har följande betydelse:

1. `form.grade.value.length > 2`
   ”Om textsträngen i textrutan med namnet `grade` har fler än två tecken”. Det är ju klart att ett tal mellan 1 och 10 måste beskrivas med maximalt två tecken.

2. `parseInt(form.grade.value) < 1`
   ”Om värdet i rutan `grade` är mindre än 1”. Funktionen `parseInt()` tar en sträng som argument och returnerar ett heltal.

3. `parseInt(form.grade.value) > 10`
   ”Om värdet i rutan `grade` är större än 10”.

4. `parseInt(form.grade.value).toString() == "NaN"`
   ”Om värdet inte är ett tal (**N**ot **a** **N**umber)”. När man använder funktionen `parseInt()` får man ibland resultatet `NaN`. Ett exempel: `parseInt(”w2”)`. Däremot ger `parseInt(”2w”)` resultatet 2.

Mellan alla dessa test har jag placerat symbolen `||` för det logiska testet ”eller”. Sammansatt blir då testets hela betydelse: ”*Om textsträngen i textrutan med namnet* `grade` *har fler än två tecken*” **eller** ”*Om värdet i rutan* `grade` *är mindre än 1*” **eller** ”*Om värdet i rutan* `grade` *är större än 10*” **eller** ”*Om värdet i rutan* `grade` *inte är ett tal*”. Det täcker de flesta knepigheter man kan få som indata. Kanske kan det första testet också få en liten förklaring. Det lyder

```
if (form.theName.value == "" && form.email.value == ""
          && form.grade.value == "")
               return false; // skicka inte tomma formulär
```

Här har jag placerat de enskilda testen mellan symbolen `&&` för logiskt ”och”. Det innebär att `if`-satsen returnerar `false` om alla villkoren samtidigt är uppfyllda. Det finns ju ingen särskild mening med att skicka ett tomt formulär. Det finns också en helt annan metod att kontrollera olika indata i formulär. Eftersom en textruta avfyrar *onChange* då värdet i den ändras kan man använda detta istället. Det kan vara en idé om man har väldigt stora formulär. Fördelen är förstås att besökaren har i färskt minne vad som borde stått i rutan (om det blev fel). Om man fångar upp ett fel direkt och låter besökaren få korrigera det omedelbart så kan formuläret upplevas mycket smidigare. För att kontrollera indata i en textruta omgående kan du skriva

```
<input type=”text” name=”email” onChange=”checkData(this)”>
```

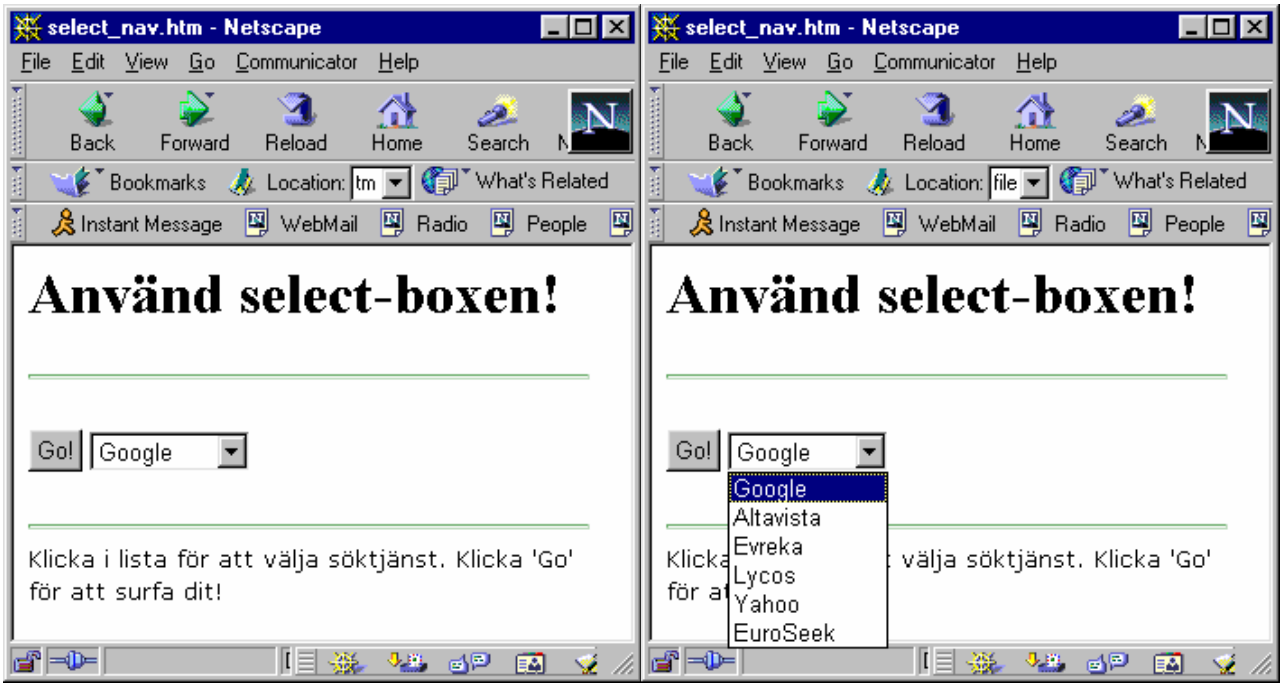
där du förstås måste definiera funktionen `checkData()`, som då kommer att få just textrutan `email` som argument. Tänk på att du måste definiera `checkData()` så här

```
function checkData(text)
{
   // satser som manipulerar med text.value
}
```

Det är `text` som kommer att få samma värde som `this` i anropet. Nyckelordet `this` refererar i det här läget till hela textrutan, så för att komma åt värdet måste du referera med `text.value`. Alternativt kan man ju anropa med `checkData(this.value)`, men å andra sidan kan man använda det förra sättet till att även kontrollera *vilken* textruta som anropat genom `text.name`. Det finns alltså ”många vägar till Rom”!

Navigera med select

På många sidor har man löst navigering, eller sökning med hjälp av `select`-boxar (eller list-boxar som de också kallas). Fördelen är att man inte behöver ta massa plats för alla de val som ingår i listan, de visas ju när man klickar i den. Jag visar nedan hur man kan tänkas använda `select`-boxen för att navigera med och sedan får du själv använda din egen fantasi för att åstadkomma det du vill ha. Först visar jag sidan och sedan koden.



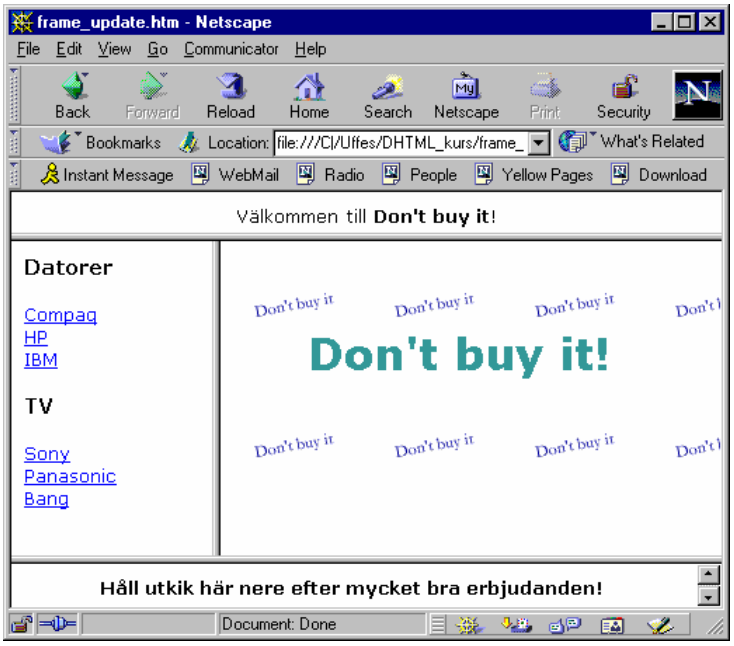

```
<HTML>
<HEAD>
<TITLE>select_nav.htm</TITLE>
<SCRIPT LANGUAGE="JavaScript"><!--

function goURL(form)
{
   var adress = form.choice.options[form.choice.selectedIndex].value;
   window.location.href = adress;
}

// --></SCRIPT>
</HEAD>
<BODY>
<H1>Använd select-boxen!</H1>
```

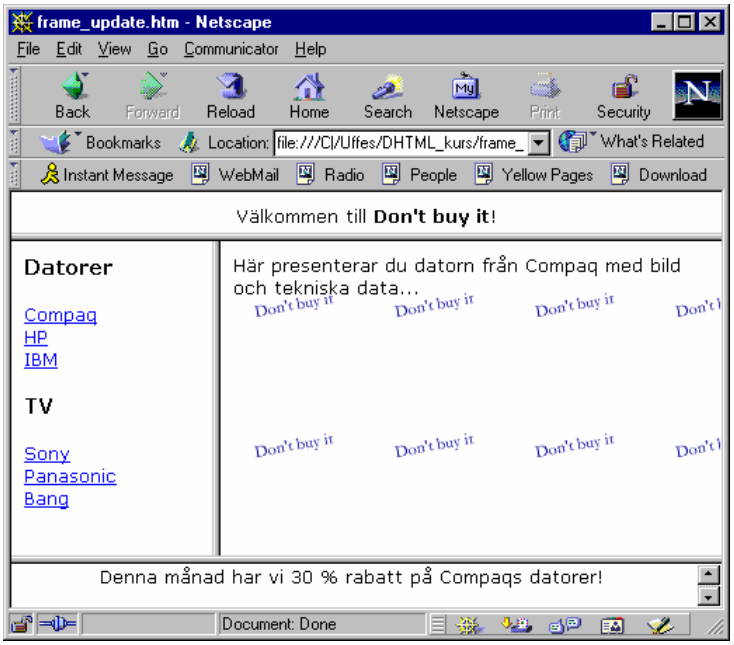
```
<HR>
<p align="right">
<FORM name="surf">
<INPUT TYPE="button" value="Go!" onClick="goURL(this.form)">
<select name="choice" size="1">
  <OPTION value="http://www.google.com">Google
  <OPTION value="http://www.av.com">Altavista
  <OPTION value="http://www.evreka.com">Evreka
  <OPTION value="http://lycos.spray.se">Lycos
  <OPTION value="http://www.yahoo.com">Yahoo
  <OPTION value="http://www.euroseek.com">EuroSeek
</select>
</FORM></p><HR>
<FONT SIZE="2" FACE="Verdana, Helvetica, Arial">Klicka i lista för att
välja söktjänst. Klicka 'Go' för att surfa dit!</FONT>
</BODY>
</HTML>
```

Hur select-boxen fungerar och hur man extraherar rätt information förklarade jag under rubriken Objektet ’Select’.

Multipla ramar

Det finns de fall då man vill uppdatera flera ramar samtidigt. Det vi hittills har lärt oss är att använda de märkord (taggar) som är standard i HTML. Bland dessa ingick att kunna definiera ett ramverk och uppdatera innehållet i en ram utifrån en länk i *en annan*. Nyckeln då var att namnge den ram man vill uppdatera (t. ex. `name = main`) och därefter i länkarna lägga till `target = main`. Men om man samtidigt vill uppdatera innehållet i två ramar samtidigt då? Ja, det finns inga taggar för sådant, men det finns ju JavaScript. Låt säga att du bygger en köptjänst och att du samtidigt som du visar upp din vara i huvudfönstret vill visa vilka eventuella erbjudanden som finns på den varan. Du bestämmer dig för att visa alla erbjudanden i en separat ram, eftersom du vill att detta hela tiden skall synas (kallas säljteknik). Då kan sidan se ut så här (skaffa dig en snyggare sida, annars kallas det dålig säljteknik..... :)
När jag sedan klickar på någon av länkarna kommer jag dels att uppdatera det stora huvudfönstret, dels miniramen längst ner där jag presenterar de aktuella erbjudanden som finns.





Den kod som genererar det här är inte särskilt komplicerad, däremot består hela tillämpningen av ganska många filer och jag visar bara den mest intressanta koden här. Ramverket har koden

```
<HTML>
<HEAD>
<TITLE>frame_update.htm</TITLE>
<script language="JavaScript"><!--

window.captureEvents(Event.ONLOAD | Event.ONUNLOAD);
window.onload = setWidth;
window.onunload = setStartWidth;

var bredd = null;
var hojd = null;

function setWidth()
{
   bredd = top.outerWidth;
   hojd = top.outerHeight;
   top.resizeTo(700, 500);
   top.moveTo(0, 0);
}

function setStartWidth()
{
   top.outerWidth = bredd;
   top.outerHeight = hojd;
}

// --></script>
</HEAD>
<frameset rows="35, *, 35">
   <frame src="frame_logo.htm" noresize scrolling="no">
   <frameset cols="150, *">
      <frame src="frame_left.htm" noresize>
      <frame src="frame_right.htm" name="main" noresize>
   </frameset>
```

```
    <frame src="frame_offers.htm" name="offers" noresize>
</frameset>
</HTML>
```

Den här filen demonstrerar hur man kan använda events lite mer sofistikerat. Man kan nämligen fånga en typ (eller flera typer) av händelse och ”peka” den händelsen till en speciell funktion. Här fångar jag *onLoad* och pekar den till `setWidth()`, som ställer fönstret till 700 * 500 px och flyttar det till skärmens övre vänstra hörn. Dessutom fångar jag *onUnload* och pekar den till `setStartWidth()` som återställer fönstret som det var innan besökaren laddade ramverket. Den här koden är inte alls nödvändig – använd den endast om du vill (det verkar dessutom inte fungera i IE). Sedan namnger jag två ramar – `main` och `offers` – och med dessa namn kan jag skripta ramarna som jag vill. Själv motorn till det hela har jag lagt i den fil som innehåller länkarna. Den sidan har koden

```
<HTML>
<HEAD>
<TITLE>frame_left.htm</TITLE>
<SCRIPT LANGUAGE="JavaScript"><!--
function loadPage(adress, index)
{
   parent.main.location.href = adress;
   parent.offers.location.hash = ”ankare” + index;
}
// --></SCRIPT>
</HEAD>
<BODY>
<FONT SIZE="2" FACE="Verdana, Tahoma, Arial">
<H4>-- Datorer --</H4>
<a href="javascript:loadPage('compaq.htm', 0)">Compaq</a><BR>
<a href="javascript:loadPage('hp.htm', 1)">HP</a><BR>
<a href="javascript:loadPage('ibm.htm', 2)">IBM</a><BR>
<H4>-- TV --</H4>
<a href="javascript:loadPage('sony.htm', 3)">Sony</a><BR>
<a href="javascript:loadPage('panasonic.htm', 4)">Panasonic</a><BR>
<a href="javascript:loadPage('bo.htm', 5)">Bang</a><BR>
</FONT>
</BODY>
</HTML>
```

Här kan du notera hur jag använder den speciella `javascript:` - URL:en för att anropa en funktion istället för att ladda ett dokument. Den funktion jag anropar laddar sedan ett dokument med ett namn som jag anger i argumentet, dessutom förflyttar den positionen för dokumentet i ramen längst ner (`offers`) till ankaret `ankare0`. Det är allt! Sidan i ramen `offers` har koden

```
<HTML>
<HEAD>
<TITLE>frame_offers.htm</TITLE>
</HEAD>
<BODY><FONT SIZE="2" COLOR="#000000" FACE="Verdana, Tahoma, Arial">
<CENTER><B>Håll utkik här nere efter mycket bra erbjudanden!</B>
<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR>
<a name="ankare0">Denna månad har vi 30 % rabatt på Compaqs datorer!
<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR>
<a name="ankare1">Denna månad har vi 10 % påslag på HP's datorer!
<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR>
<a name="ankare2">Om du köper en dator från IBM så borde du få en från
Compaq gratis !!!
```

```
<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR>
<a name="ankare3">När du köper en widescreen TV från Sony får du ett
hemmabiosystem på köpet! Hugg!
<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR>
<a name="ankare4">En Panasonic vill du inte ha!
<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR>
<a name="ankare5">Varför köpa danskt?
<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR></CENTER></FONT>
</BODY>
</HTML>
```

Här lägger jag in massor med radbrytningar för att belysa endast erbjudandet vid respektive ankare. Enkla tricks funkar ofta rätt bra...

Animering med JavaScript

Alla känner nog till att man kan skapa animerade gif:ar till webbsidor, men att man kan skapa animeringar med jpg-bilder visste ni kanske inte. Nu är det förmodligen inte så väldigt användbart, men ändå...
I det här exemplet tillverkar jag en liten animering med tre jpg-bilder. Jag visar inte sidan här (känns tämligen meningslöst – jag kommenterar istället), men koden är i alla fall som följer:

```
<HTML>
<HEAD>
<TITLE>js_animering.htm</TITLE>
<SCRIPT LANGUAGE="JavaScript"><!--
nr = 0;
function next()
{
  nr = (nr + 1) % 3; // stega en bild i taget och börja om efter 3 steg
  document.images[0].src = "google_off" + nr + ".jpg";
}
// --></SCRIPT>
</HEAD>
<BODY><center>
<H1>Animering med JavaScript!</H1><HR>
<img src="google_off0.jpg" onLoad="setTimeout('next()', 1000)">
</center>
</BODY>
</HTML>
```

Jag använder mig av `image`-objektets *onLoad* hanterare. *Varje gång* en bild laddats kommer detta event att avfyras, jag fångar *onLoad* och anropar funktionen `next()`. När jag gör anropet sätter jag även en timeout på 1000 millisekunder (1 sekund), annars kommer animeringen att gå så fort att våra ögon inte kommer att uppfatta den. Inuti funktionen `next()` ökar jag den globala variabeln `nr` ett steg, därefter låter jag modulo-operatorn verka på den. Det betyder att `nr` endast kan anta värdena 0, 1 och 2 (eftersom detta är de enda möjliga resterna vid division med 3). Därefter låter jag den första (och enda) bilden i sidan få ny källfil med

```
document.images[0].src = "google_off" + nr + ".jpg";
```

Eftersom `nr` endast kan anta värdena 0, 1 och 2, måste bilderna vara sparade som `google_off0.jpg`, `google_off1.jpg` och `google_off2.jpg`. Eftersom *onLoad* avfyras varje gång en bild laddats kommer hela förloppet att upprepas tills man laddar en annan sida.

Ulf Tornert
ulftt@gy.varmdo.se

Pop-up fönster

Jag visade under rubriken Objektet 'window' hur man öppnar nya webbläsarfönster, sk pop-up fönster. Där hittar du det mesta av det du behöver för att hantera dessa.

Skaka av ramverk

Ett vanligt trick är att se till så att inget yttre ramverk laddar dina sidor. Detta är särskilt användbart om dina egna sidor är uppbyggda av ramar, eftersom det blir så ”plottrigt” med ramar i ramar i ramar... Allt du behöver göra är att lägga till en liten kodsnutt i `<head>`-sektionen i den fil som definierar ditt ramverk. Kodsnutten är

```
if (self.location.href != top.location.href)
  top.location.href = self.location.href;
```

(inuti `<script>`-taggar förstås!). Dvs om URL:en till det aktuella ramverket inte överensstämmer med URL:en i huvudfönstret, så sätter man just denna till adressen till ramverket. Mycket enkelt!

Kontrollera att en användare alltid laddar ditt ramverk

Det omvända är lika enkelt. Låt säga att en besökare har sökt upp dina sidor med någon sökmotor på Internet. Denne råkar då ladda en fil som egentligen är tänkt att ingå i ett ramverk. Då kan flera saker inträffa – dels missar besökaren förmodligen både information och logik över hur man navigerar på din sajt, dels kommer skriptfel att uppstå om sidan exekverar skript med referenser till ramar som inte finns. Inte bra! Då är det en god idé att se till så att alla ingående sidor på din sajt (som skall gå under ditt ramverk) använder följande kodsnutt:

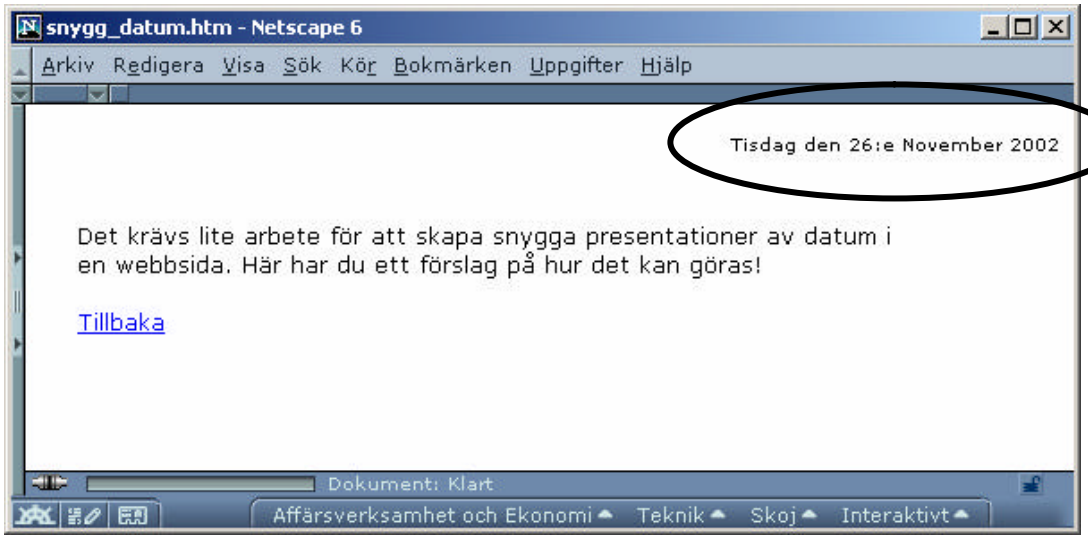
```
if (top.location.href == self.location.href)
  top.location.href = ”adressen till ditt omgivande ramverk”;
```

Det problem du kan få är förstås att ramverket är definierat för att ladda vissa sidor när det startar, och den sida som gör kontrollen ovan är förmodligen inte en av dem. Här gäller det att meddela besökaren vad som händer, eller så får man utöka sina kunskaper med lite asp, php, jsp eller cgi ✍.

Snygg datum

Att skapa snygga presentationer av datum och tid i en webbsida kräver lite arbete. Jag visar här ett förslag på hur det kan göras. Sidan som genereras får följande utseende:

Ulf Tornert
ulftt@gy.varmdo.se



Koden till sidan är

```
<HTML>
<HEAD>
<TITLE>snygg_datum.htm</TITLE>
<SCRIPT LANGUAGE="JavaScript"><!--

  var veckoDag = new Array(7);
  veckoDag[0] = "Söndag";
  veckoDag[1] = "Måndag";
  veckoDag[2] = "Tisdag";
  veckoDag[3] = "Onsdag";
  veckoDag[4] = "Torsdag";
  veckoDag[5] = "Fredag";
  veckoDag[6] = "Lördag";

  var manad = new Array(12);
  manad[0] = "Januari";
  manad[1] = "Februari";
  manad[2] = "Mars";
  manad[3] = "April";
  manad[4] = "Maj";
  manad[5] = "Juni";
  manad[6] = "Juli";
  manad[7] = "Augusti";
  manad[8] = "September";
  manad[9] = "Oktober";
  manad[10] = "November";
  manad[11] = "December";

  function fixYear(ar)
  {
    if (navigator.appName == "Netscape")
    {
      ar = parseInt(ar - 100);
      if (ar < 10)
        ar = "200" + ar;
      else
        ar = "20" + ar;
      return ar;
    }
```

```
      else
         return ar;
   }

// --></SCRIPT>
</HEAD>
<BODY>
<script language="JavaScript"><!--
   var idag = new Date();
   dag = idag.getDay();
   datum = idag.getDate();
   manadNr = idag.getMonth();
   ar = idag.getYear();
   var counted;
   if (datum < 3 || datum > 30)
      counted = ":a";
   else if (datum > 20 && datum < 23)
      counted = ":a";
   else counted = ":e";
   datum = datum + counted;
   output = "<P ALIGN='right'><FONT SIZE='1' FACE='verdana, arial, sans-
serif'>" + veckoDag[dag] + " den " + datum + " " + manad[manadNr] + " " +
fixYear(ar) + "</FONT></P>";
   document.write(output);
// --></script>
<table border="0" cellspacing="20" width="500">
<TR>
   <TD><P><FONT SIZE="2" FACE="Verdana, Tahoma, Arial">Det krävs lite
arbete för att skapa snygga presentationer av datum i en webbsida. Här
har du ett förslag på hur det kan göras!<P><a
href="../index.htm">Tillbaka</a></FONT></TD>
</TR>
</table>
</BODY>
</HTML>
```

Det här exemplet är en utvidgning av det som finns under rubriken Objektet ’Date’. Jag börjar med att definiera två vektorer (`Array`-objektet), där jag lagrar namnen på veckodagarna och månaderna i det svenska språket. Sedan skapar jag en funktion för att justera utseendet på årtalet. Sedan tar jag ett snapshot av datorns systemklocka, lagrar detta i variabeln `idag` och extraherar dagen, datum, månad och år från denna. Då får jag alla ’rådata’ som jag behöver för att presentera rätt datum – resten är kosmetika. Satserna

```
var counted;
if (datum < 3 || datum > 30)
   counted = ":a";
else if (datum > 20 && datum < 23)
   counted = ":a";
else counted = ":e";
```

ser till så att `counted` kommer att vara antingen ’:a’ eller ’:e’, beroende på vilket datum det är. Om till exempel `datum = 21`, kommer jag att visa `21:a` i sidan. Satsen `datum = datum + counted;` kommer att slå ihop heltalet i `datum` med tecknet i `counted`. Resultatet blir en sträng. Sedan definierar jag innehållet i en variabel (`output`) som ger själva datumutskriften:

```
output = "<P ALIGN='right'><FONT SIZE='1' FACE='verdana, arial, sans-
serif'>" + veckoDag[dag] + " den " + datum + " " + manad[manadNr] + " " +
fixYear(ar) + "</FONT></P>";
document.write(output);
```

Här ser du ett exempel på hur man kan sätta samman (konkatenera) ganska komplicerade uttryck till en enda sträng. Lägg märke till att jag konkatenerar strängar med variabler. Var noga med hur du använder dubbla citationstecken (”) och enkla apostrofer (’). Du får bara nästla dem inuti varandra så länge det blir matchande par. Annars kan inte JavaScript-tolken avgöra vad som skall bli den slutgiltiga strängen. Nu har jag inte tagit med någon presentation av tid i sidan, men det kan du få göra på egen hand.

Övningsuppgifter

I den här delen får du möjlighet att själv skapa några JavaScript för tränings skull. Skripten kanske inte har särskilt stora användningsområden, men de syftar till att lära dig och ge dig trygghet i att skapa egna, fungerande JavaScript. Tänk på att olika webbläsare tolkar JavaScript lite olika, dessutom finns det olika dialekter och versioner av språket. Det förvärrar situationen avsevärt när man provar ut ett skript. Om du får problem med ett skript, prova då med en annan browser – det kan hjälpa. Om du fastnar kan du få mycket hjälp genom att skriva **`javascript:`** i adressfältet i NN. Du får då en JavaScript-konsol som ger dig information om syntaktiska felaktigheter. Den protesterar däremot inte mot logiska felaktigheter, dessa måste du kontrollera själv. Om du får problem med exempelvis funktionsanrop, eller med att variabler inte ger rätt värden etc, kan du med fördel använda `alert()` för att visa värdet av en variabel vid ett givet tillfälle, exempelvis då en funktion startar eller slutar. Om du fastnar helt och hållet kan du hitta alla filer på min hemsida (**`http://hem.passagen.se/ulftt`**). Lägg till exempelvis **`ovnuppg_1.htm`** för att ladda koden för den sidan. Men gör dig själv en tjänst – tjuvtitta inte förrän du försökt själv! Risken är stor att du inte lär dig så mycket då. Lycka till!

1. **`ovnuppg_1.htm`**
   Skriv ett skript som kontrollerar vilken webbläsare en besökare använder. När du vet vilken det är skall du påminna denne via en `alert`-box, ungefär som nedan. Du får utgå från att

   besökaren använder antingen NN eller IE. *Tips*: Använd `navigator.appName`.

2. **`ovnuppg_2.htm`**
   Utöka övningen ovan till att testa om webbläsaren antingen är NN eller IE, om det inte är någon av dem skall du ge meddelandet ”Du använder inte Netscape eller Explorer” i en `alert`-box. *Tips*: Använd nästlade if-satser.

3. **`ovnuppg_3.htm`**
   Nu utökar vi övningen ovan till att testa både namnet och versionsnumret på den aktuella webbläsaren. Om webbläsaren är NN eller IE av versionsnummer 4 eller större, skall besökaren få meddelandet ”Din webbläsare hanterar mina skript”. Men om webbläsaren inte

motsvarar kraven skall meddelandet ”Uppgradera din webbläsare” ges. Du väljer själv om du vill använda en `alert`-box, eller om du vill skriva meddelandet direkt i sidan. *Tips*: Använd `parseInt(navigator.appVersion)` för att testa versionsnumret. För att effektivisera testerna bör du använda de logiska operatorerna, vilka utför de logiska testerna. Ett pseudokod-exempel

```
if (webbläsaren == ”Netscape” && versionsnumret >= 4)
```

4. **`ovnuppg_4.htm`**
   Tillverka en for-slinga som skriver ut en sträng i sidan 5 gånger med radbrytning efter varje rad. Börja varje rad med numrera den. *Tips*: Använd `document.write()` och konkatenera (slå samman) uppräkningsvariabeln i for-slingan med din textsträng.

5. **`ovnuppg_5.htm`**
   Gör samma sak som ovan men med while-slingan. Skapa en uppräkningsvariabel utanför slingan, som sedan räknas upp inuti den.

6. **`ovnuppg_6.htm`**
   Tillverka ett formulär som innehåller två textrutor och en knapp. Du skall kunna kopiera innehållet i den första textrutan till den andra genom att trycka på knappen. *Tips*: Använd knappens *onClick* för att anropa en funktion som du själv skapar. Tänk på att innehållet i en textruta nås genom `document.formularNamn.textrutaNamn.value`.

7. **`ovnuppg_7.htm`**
   Skapa en sida där en besökare kan bestämma själv vilken bakgrundsfärg denne vill ha på sidan. Tillverka ett formulär där besökaren kan skriva i antingen en färgkod eller ett färgnamn. Bakgrundsfärgen ändras när man klickar på en knapp i formuläret. *Tips*: Du ska manipulera med `document.bgColor`. Du behöver inte kontrollera om besökaren knappar in nonsens-data!

8. **`ovnuppg_8.htm`**
   Omforma uppgiften ovan så att en besökare kan skriva i en URL i textrutan. När denne klickar på knappen ska denna öppnas i browsern. *Tips*: Lägg strängen i textrutan till `window.location.href`.

9. **`ovnuppg_9.htm`**
   Ändra nu ytterligare på uppgiften så att den adress besökaren skriver in öppnas i ett nytt fönster. Nu skall formuläret även innehålla en knapp med vilken man kan stänga det nya fönstret med. *Tips*: Skapa en global variabel med namnet `newWin`. När du öppnar det nya fönstret exekverar du satsen `newWin = window.open(”adressen”, ””);` Därefter kan du använda `newWin.close()` för att stänga det igen.

10. **`ovnuppg_10.htm`**
    Skriv en liten sida som låter hälsa ”Sidan är laddad!” då den är laddad i sin helhet. *Tips*: Använd *onLoad*.

11. **`ovnuppg_11.htm`**
    Gör en sida som multiplicerar två tal från varsin textruta i ett formulär. Den funktion du skapar skall alltså ta två argument (parametrar). *Tips*: Använd `parseInt()`.

12. **`ovnuppg_12.htm`**
    Gör samma sak som ovan, men addera de två talen istället. *Tips*: Här måste du skicka de två värdena *efter* det att du först omvandlat dem till heltal med `parseInt()`. JavaScriptbug? Både NN och IE kräver detta. Prova vad som händer om du inte gör det!

13. **`ovnuppg_13.htm`**
    Skapa en sida innehållande en länk. Den sida som länken refererar till (här väljer du vad som helst) skall öppnas i ett nytt fönster, där locationbar, menubar, toolbar och scrollbar skall vara synliga. Fönstrets initiala storlek skall vara 400 * 300 px, det ska dessutom gå att ändra på denna genom att ”klicka och dra”. *Tips*: Kolla under rubriken Objektet ’window’ i boken.

14. **`ovnuppg_14.htm`**
    Prova att kopiera delar av en sträng med med funktionen `substring()`. Låt användaren själv skriva in strängen och start- och stopindex. Då skapar du enklast ett formulär med tre textrutor för inmatning och en knapp för att exekvera funktionen. Du kan skicka ut resultatet i en `alert`-box eller i en fjärde textruta i formuläret. Kontrollera att stopindex inte *är större än* antalet tecken i strängen. *Tips*: Kom ihåg att JavaScript indexerar både vektorer och strängar från 0 och uppåt. Index 3 motsvarar alltså det *fjärde* tecknet i en sträng. I satsen

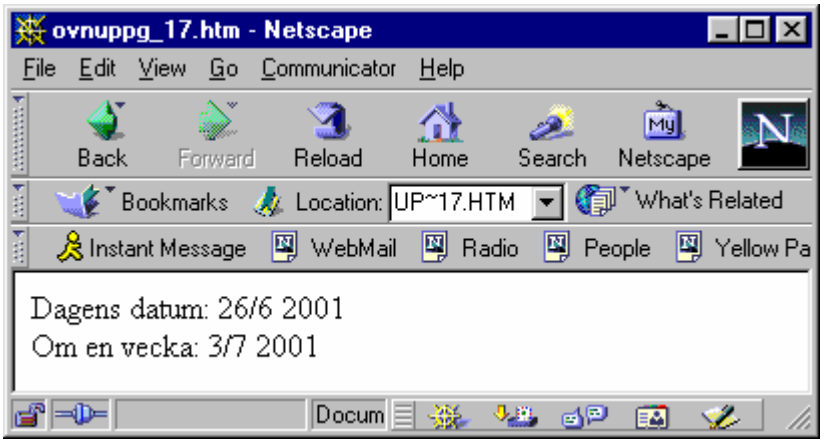
    ```
    var name = myString.substring(0, 4);
    ```

    kommer `name` att få tecknen i `myString` från index 0 t.o.m. 3, dvs det tecknet som står omedelbart före det som anges i parentesen!

15. **`ovnuppg_15.htm`**
    Jämför om två strängar är lika genom att anropa `toLowerCase()` eller `toUpperCase()` på dem. Låt besökaren fylla i två textrutor, därefter får denne trycka på en knapp varpå jämförelsen utförs. Ge besökaren en bekräftelse på resultatet. *Tips*: Använd `onFocus="this.blur()"` om du presenterar resultatet i en textruta. Då kan ingen komma åt innehållet i den.

16. **`ovnuppg_16.htm`**
    Skapa ett `Date`-objekt och visa mha detta dagens datum, månad och år. Försök återskapa ungefär sidan nedan. *Tips*: Tänk på att `getDate()` returnerar ett tal mellan 1 och 31, `getMonth()` ett tal mellan 0 och 11, slutligen returnerar NN4:s version av `getYear()` det aktuella årtalet minus 1900.

17. **`ovnuppg_17.htm`**
    Utöka uppgiften ovan genom att skapa ytterligare en variabel som du kan kalla `nextWeek`. Denna skall visa datum om precis en vecka. *Tips*: Först måste du avläsa tidstämpeln på den variabeln som du nyss skapat (jag kallar den `idag`) – det gör du med `getTime()`. Du måste sedan lägga till det antal millisekunder som en vecka motsvarar i variabeln nextWeek – det gör du med `setTime()`. Rätt antal millisekunder på en vecka är 1000*3600*24*7. Presentera båda datumen i samma format (som ovan).



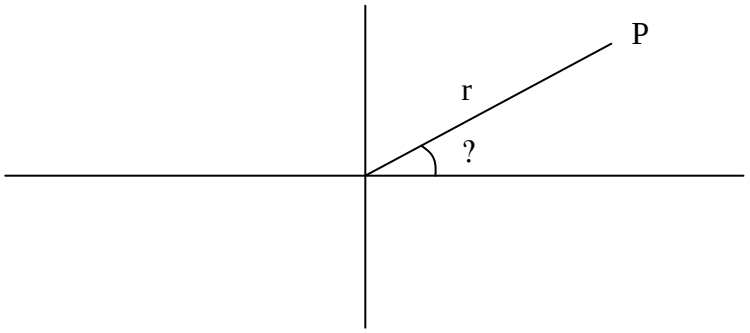
18. **`ovnuppg_18.htm`**
    Skapa ett formulär innehållande fem radioknappar. I `value`-attributet till varje knapp anger du en URL (lokal eller global). Skriv en liten förklaring till vart varje knapp leder på Internet och låt besökaren surfa till en annan sida genom att klicka i någon av knapparna. *Tips*: Här tycker jag att while-slingan passar bra. Använd den för att testa huruvida en knapp är nedtryckt eller inte. Om en knapp trycks ned kommer medlemmen `checked` att få värdet `true`. Tänk på att samtliga radioknappar måste ha samma namn i name-attributet, annars fungerar det inte. Testet i while-slingan kan då utformas så här

    ```
    while (document.forms[0].go[i].checked != true)
    ```

    där `go` är namnet på radioknapparna och `i` är en uppräkningsvariabel.

19. **`ovnuppg_19.htm`**
    Skapa en sida där besökaren kan ”slå en tärning”. Besökaren kan bestämma ett tal mellan ett och sex, och sidan håller reda på hur många gånger det antalet prickar dyker upp. Efter varje kast skall besökaren kunna se hur många kast han gjort, hur många gånger hans värde erhållits och kvoten mellan antalet träffar och antalet kast. *Tips*:

20. **`ovnuppg_20.htm, snurren.htm`**
    Ett exempel för matteintresserade! Den här övningen innehåller en hel del matematik (lite för att demonstrera objektet `Math`), så om du vill kan du hoppa över denna. Det du ska skapa nu är en sida som öppnar ett pop-up fönster med storleken 100 * 100 px, och som skall placeras *mitt* på skärmen (du får utgå från antingen upplösningen 800 * 600 eller 1024 * 768). Den sida som visas i pop-up fönstret innehåller endast en knapp och när man klickar på den kommer fönstret att förflytta sig i en spiral utåt och slutligen försvinna ur sikte. När fönstret har försvunnit utanför skärmkanten ska du återföra det till mitten, där det började. *Tips*: För att få fönstret till mitten av skärmen då upplösningen är 800 * 600 kan du köra satsen `newWin.moveTo(400 - 50, 300 - 50)`, eftersom det är 400 px från vänster till mitten och halva fönstret är 50 px brett (på samma sätt i höjdled). Utgångspunkten är alltså (350, 250) och sedan gör vi förflyttningar på fönstret relativt denna punkt, dvs vi upprepar satsen `newWin.moveTo(350 - x, 250 - y)`, för `x` och `y` som skall bestämma koordinaterna för en spiral utifrån mitten. Hur åstadkommer man en spiral? Tänk er att vi sätter ett koordinatsystem i mitten av skärmen, så att origo sitter i punkten (350, 250).



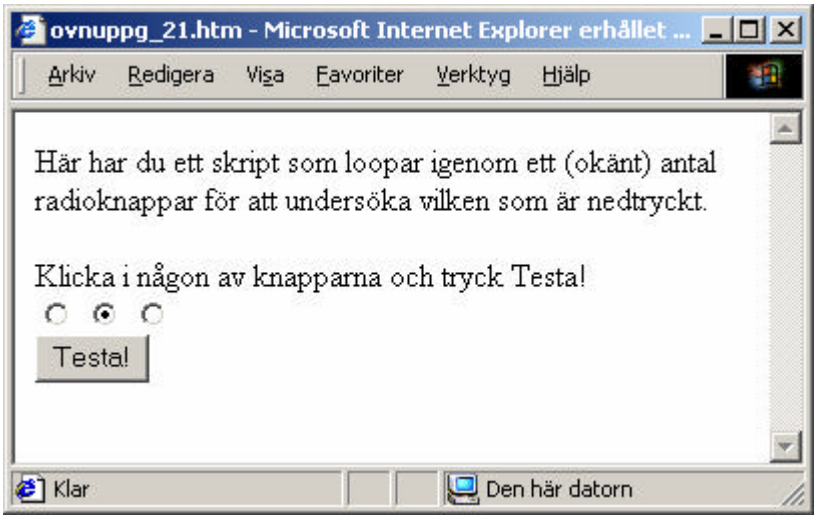
Punkten P:s koordinater kan beskrivas med (350 – x, 250 – y), där $x = r \cdot \cos?$ och $y = r \cdot \sin ?$. Det enda jag behöver göra är att öka `r` och ? så att P kommer att beskriva en spiral från origo. Om jag skapar en slinga där jag för varje `r` och ? beräknar `x` och `y`, och därefter flyttar fönstret med `newWin.moveTo(350 - x, 250 - y)`, så får jag ju den önskade spiralen! Avbryt slingan då radien är såpass stor att fönstret hamnar utanför skärmkanten. Min slinga ser ut så här

```
while (r < max)
{
   x = r * Math.cos(teta * Math.PI/180);
   y = r * Math.sin(teta * Math.PI/180);
   self.moveTo(350 - x, 250 - y);
   teta += 10;
   r += 10;
}
```

där `r`, `max`, och `teta` är *globala* variabler (skapade utanför while-slingan). Variablerna `r` och ? startar båda på 0, sedan ökas de med värdet 10. För `r` betyder det att radien ökar med 10 px, och för ? betyder det att vinkeln ökar med 10?. Men `Math`-objektet räknar inte med grader utan

radianer, vilket gör att jag måste räkna om gradantalet till denna enhet. Det gör jag genom att först multiplicera med ? (`Math.PI`) och sedan dividera med 180. När `r` överstiger `max` kommer slingan att sluta upprepas och det är dags att återföra fönstret till utgångspunkten!

21. **`ovnuppg_21.htm`**
    En lite svårare uppgift! Denna går ut på att ta reda på vilken radioknapp en användare har tryckt ner. I många formulär på webben har man möjlighet att välja exempelvis preferenser vad gäller musik, teater, film etc. Det är din uppgift som webbdesigner att ta hand om formuläret och vidarebefordra informationen för bearbetning. Skapa ett formulär med en grupp om tre radioknappar och en vanlig knapp. Man skall nu kunna välja en radioknapp och få en kvittens på vilken det är man tryckt ner. Kontroll sker med den vanliga knappen. Sidan kan se ut så här:



Jag hoppas att den här kursen har gett dig lite nya kunskaper. Min förhoppning är att du åtminstone skall ha fått självförtroende nog att skapa (eller knycka) några *enkla och användbara* skript, som du kan använda på dina sidor. Mycket av det som finns i det här häftet är svårt att förstå om man inte har programmerat förut, och du skall inte känna dig nedslagen om du upplever det så. Det här häftet är skapat för att ge *också dem* som har erfarenhet av programmering, eller som har stort intresse, något att bita i. Jag önskar dig lycka till!

Ulf Tornert