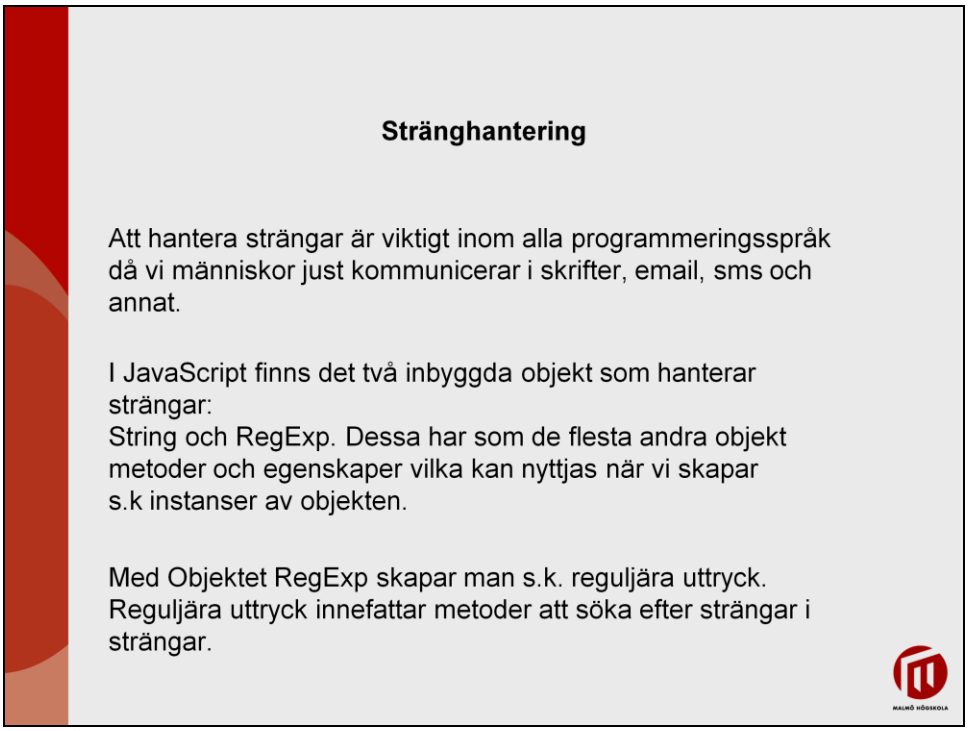
Föreläsniningsmaterialet är baserat på material från John Niubo, uppdaterat av Anton Tibblin.



Mer om Strängar:

Det finns mängder av inbyggda metoder till objektet String().

Ex: strangnamn**.toLowerCase()** som återger strängen omvandlad till gemener. För resterande metoder hänvisar vi till litteraturen eller listor över metoder(API) på Internet.

Mer om Reguljära uttryck:

Det fina med reguljära uttryck är att man kan söka efter vissa definierade saker i en text-sträng. Jämför t.ex när du söker i windows efter alla filer som du gjort innehållande namnet examen, då kan du söka efter dem med söksträngen *examen*.*

Reguljära uttryck - Regular Expressions

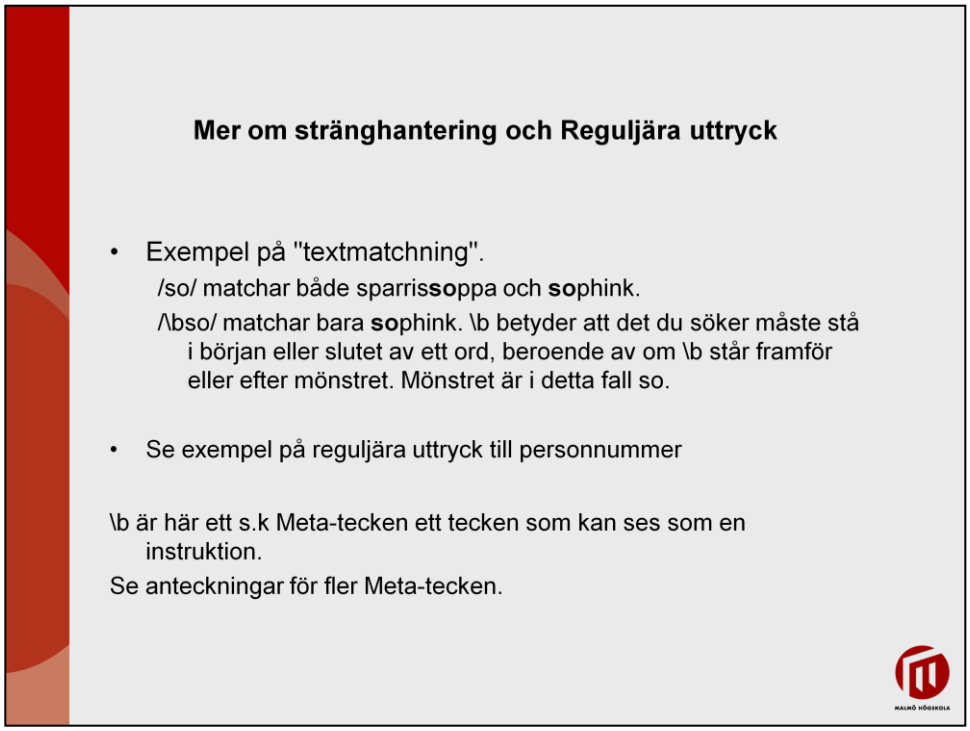
- Kan användas för att söka i en sträng eller ett *mönster* i en sträng
- Härstammar från UNIX, återfinns i linux-editorer (vi, emacs, sed), i shell-kommandot egrep
- Används flitigt i t.ex
  Awk, C#, JavaScript, Perl, Python, Tcl, Visual Basic och ASP/**PHP**
- Reguljära uttryck är alltså inte bundna till ett speciellt språk.
- Används primärt till att kontrollera formulär på klientsidan.
  - Det är effektivare att kontrollera ett formulär m.a.p inmatning direkt på klientsidan med JavaSript än att skicka det via Internet till ett serverbaserat script.
- Kan också användas till:
  - Söktjänster på egna webbplatser.
  - Kontroll av IP-nummer. etc. etc.
- Reguljära uttryck består av:
  - "Vanliga tecken", dvs a-z och 0-9.
  - Metatecken - speciella tecken som är tilldelade speciell betydelse.

Mer om stränghantering och Reguljära uttryck

- Mer om Reguljära uttryck
  -Det finns 2 sätt att skapa instanser av objektet RegExp().
  var variabelnamn = new RegExp("reguljärt uttryck"); samt
  var variabelnamn=/reguljärt uttryck/;
  (i det senare fallet skall inte citationstecken användas.)

- Om man t.ex vill söka efter något genom en hel sträng (s.k global sökning) kan man använda diverse inbyggda funktioner hos objektet RegExp.

- Generella reguljära uttryck. Ett sätt att söka efter delar av strängar och s.k mönster är om vi t.ex letar efter "op", vilket kan finnas i strängarna "opponera" och "operation". För att genomföra sådan sökningar behöver man s.k ***Metatecken.*** Dessa representerar andra tecken (jfr jokertecken, ? och *, som används för att representera ett resp. flera tecken, vilka som helst). Metatecken används också för att ange (ge instruktioner) var det reguljära uttrycket ska matcha strängen.



Andra intressanta metatecken:

\s	mellanslag. /ap\ssvans/ matchar "ap svans" men inte "ap-svans"
\d	siffra. /Hus\s\d/ matchar "Hus 6", "Hus 2" etc
\w	bokstav, siffra eller _ (underscore). (de tecken som är giltiga som variabelnamn). /K\w/ matchar KY, K_, K1 men inte K¤ eller K-.

Byter man ut de små bokstäverna till stora (\S, \D och \W) får metatecknerna motsatt betydelse, dvs \D matchar allting som *INTE* är siffror

För att sen bestämma hur många av olika tecken som tillåts använder man andra tecken tillsammans med metatecknet.

\d*	noll eller fler siffror
\d?	noll eller en siffra
\d+	en eller flera siffror
\d{MIN}	exakt MIN stycken siffror
\d{MIN,}	minst MIN stycken siffror
\d{MIN,MAX}	minst MIN och högst MAX stycken siffror

t ex en personnummerkoll, /\d{6}\d{2}?-\d{4}/

Strängmetoder som använder reguljära uttryck

- Det finns ett antal strängmetoder som använder reguljära uttryck.
- Dessa heter och gör:
  - search (reguttryck) Hittar startpositionen av reguttryck
  - match (reguttryck) Söker efter reguttryck och returnerar en sträng som innehåller det sökta "reguttryck" till en vektor (array).
  - replace (reguttryck) Ersätter reguttryck med valfri sträng.
  - split (reguttryck)

    Delar upp en textsträng till en vektor (array) av strängar. Uppdelnings förfarandet d.v.s var vi bestämmer oss att klippa strängen beror på reguttryck.

  Till alla dessa metoder har vi s.k flaggor som vi kan lägga till för att ändra sökmönstret:
  - i ignorera gemen/versal skillnaden.
  - g träffa alla förekomster av det reguljära uttrycket i stängen.
  - m träffa över flera linjer i strängen.

Exempel - search

- Alla objekt av strängtyp i Javascript har funktionen search().
  - Inargument: reguljärt uttryck
  - Returvärde: Position där *första* matchningen påträffades. Om ingen matchning gjordes returneras -1.
- Exempel:

Exempel på search

```
var s = prompt("Ange namn", "");
if (s.search("Nils") == -1) {
   alert("Det var inte Nils!");
} else {
   alert("Det var Nils!");
}
```

- Obs att endast imatningen Nils ger utskriften "Det var Nils!"
- Inmatningen nils ger "Det var inte Nils!"

- JavaScript är alltså känsligt för gemena och versala bokstäver d.v.s Case-sensitive.

Exemplet finns med på kurssidan och heter: reguttryck_ex1.htm

search
-Flaggan Case Insensitive

Flaggan **i** gör uttrycket *okänsligt* för stora och små bokstäver

- Fungerar ej på Å, Ä och Ö.
- Flaggan skall sättas *efter* slutmarkör av en sträng.

Ex:

```
s.search("Nils","i");
s.search(/Nils/i);
```

Detta är de två sätten att ange ett reguljärt uttryck direkt i search funktionen som en Inparameter.

- Obs att alla kombinationera av bokstäverna till Nils ger utskriften "Det var Nils!"

Exemplet finns med på kurssidan och heter: reguttryck_ex2.htm

Match

- Match söker efter en förekomst av ett reguljärt uttryck och returnerar mer data:
  - input
  - index
  - lastIndex
  - 0, 1, 2, 3...

Exempel match

```
var a = "aaabbbccc";

i = a.match(/b{3}/);

if (i != null) {
  document.write("input: " + i.input +
            "index: " + i.index +
            "lastIndex: " + i.lastIndex +
            "element 1: " + i[0] );

}

ger:
      input: aaabbbccc
      index: 3
  lastIndex: 6
  element 1: bbb
```

Flaggan g

- Flaggan g - global innebär att en sökning repeteras tills strängen är slut - även om en matchning redan skett.
- Ex, att göra funktionen *s.split(",")* med reguljära uttryck:

```
var s = "erik,pelle,adam";
var arr = s.match(/([^,]*)/g);
```

Metoden Replace

- Replace kan
  - byta ut ord i sträng
  - ta bort vissa ord/tecken ur sträng
  - Göra länkar av text
- Ex: Att byta ut ordet 'myra' mot 'elefant'
- ```
  var s = 'Titta, en myra, det är ett stort ' +
          'djur! Jag vill klappa myraen!';
  alert( s.replace(/myra/g , "elefant") );
  ```


  ```

Replace - rensa farliga tecken

Vi ska använda strängen s1 i eval, men vill
försäkra oss om att den bara innehåller matematiska tecken 0-9,
+, -, * och /.

```
var s1 = 'Nu ska vi se, vad blir ' +
         '100 * 80 / 10 + 10 månne?';
var s2 = s1.replace(/[^0-9\+\-\*\/]/g , "");
var s3 = eval ( s2 );
document.write( s2 + " blir " + s3 );
```

```
ger: 100*80/10+10 blir 810
```

m beskriver början av ett reguljärt uttryck med /

anger en parentes eftersom vi vill gruppera mönstret vi letar efter.

http: letar vi efter men också två / efter detta. För att ange / utan att få det reguljära uttrycket att tolkas fel så måste vi

markera det med \. Detta görs två gånger eftersom det är två st tecken som ej ska tolkas.

[^\n]* betyder att teckenmängden [] som INTE börjar med en ny rad och mellanslag 0 eller flera gånger.

Detta körs genom hela texten /g

Paranteser

- Parenteser
  - () Grupperar mönster (används bl a ihop med sök/utbyt-operationer)
  - {n, m} Anger hur många gånger utrycket till vänster om klammern ska skall matchas.
  - [ ] Anger teckenmängd

Exempel på teckenmängd

- En teckenmängd sätts mellan [].

Exempel:
Användaren ska mata in **KOD** följt av 2 siffror.

```
var s= prompt("ange KOD följt 2 siffror","");
if (s.search(
        /KOD[\s][0123456789][0123456789]/)==-1) {
    alert("FEL!");
} else {
    alert("OK!");
}
```

\s står för blanksteg tab eller newline.

==-1 är samma sak som false. Exemplet kan göras enklare tänk på hur...

Se exempel reguttryck_ex3.htm på kurssidan.

Paranteser - "måsvingar"

- Ett bättre sätt att ange **[0123456789]** är att skriva **[0-9]**. Istället för att upprepa satsen 2 ggr kan man sätta **{n, m}** efter.
  - n = MINSTA antalet repetioner
  - m = MAX antal repetioner
  - Om ,m utelämnas gäller *exakt* n ggr
  - Om m utelämnas (men , finns kvar) gäller *minst* n ggr.

- exempel: **s.search(/KOD[0-9]{2}/)**
  här ska det alltså komma *exakt* 2 siffror
  efter ordet KOD för att vi ska få träff

Repetition av teckenföljd

- Vanliga parenteser () grupperar en teckenföljd.
- Exempel

```
var s = prompt("Skriv: hipp hipp hurra, hurra, hurra","");
<script type="text/javascript">
var s = prompt("Skriv: hipp hipp hurra, hurra, hurra","");
if (s.search(/[\s*hipp](\shurra){3,3}/)==-1)
   {alert("FEL!");}
 else
   {alert("OK!");}
</script>
```

Reguljärt uttryck, omgärdat av parentes.
s står för space (mellanslag) eller tab
s* står för mellanslag 0 eller flera gånger

Kör exemplet reguttryck_ex4.htm på kurssidan.

Fler sätt att ange antal

- {n,m}
- {n}
- * (0 eller fler ggr)
- + (1 eller fler ggr)
- ? (0 eller 1 gång)

Fler metatecken

- ^ Matchar radbörjan (ankare)
- $ Matchar radslut (ankare)
  ex:
  - `s.search(/^nisse$/) == 0` är ekvivalent med
  - `s == "nisse"`
- . Matchar *exakt* ett tecken
- | Matchar föregående eller nästkommande text ex:
  - `s.search(/^nisse$|^erik$|^lennart$/) == 0` är ekvivalent med
  - `(s == "nisse") or (s == "erik") or (s == "lennart")`

Se exempel reguttryck_ex5.htm på kurssidan.

Inverterare i teckenmängd

- Då en teckenmängd definieras inom [och] ,betyder ^ *INTE eller negering.*
  *D*et är således skillnad på om ^ sitter innanför en hakparentes eller ej.
- Följande uttryck betyder: *Hela strängen s måste bestå av en eller fler siffror. Inget annat!* `s.search(/^[0-9]+$/)`
- Om vi istället använder ^ som negering blir det: *Hela strängen måste innehålla minst ett tecken, och inga tecken får vara numeriska!* `s.search(/^[^0-9]+$/)`

Formulär med validering

- Det vanligaste området för reguljära uttryck i JavaScript är att validera inmatning i ett formulär ***innan*** användaren får skicka information till webbservern.
- Exempel på en sida som ber användaren fylla i sitt mobiltelefonnummer.
- Kriterier:
  - Får bara innehålla numeriska tecken
  - Måste börja med 07
  - Måste vara exakt 10 tecken förutom mellanslag eller bindestreck.

```
<form name="form1" onSubmit="return kontrollera(); ">
    Telefonnr: <input type=text name=telnr>
</form>
```

Kontrollera funktionen ser vi på nästa bild.

Formulär med validering
-Funktionen *kontrollera*

```
<script type="text/javascript">

function kontrollera() {
   var c = document.form1.telnr;
   if (c.value == "") {
     c.focus();
     alert("Du måste mata in ditt mobiltelefonnr!");
     return false;
   }
   if (c.value.search(/^07/)==-1) {
     c.focus();
     alert("Ditt mobilnr måste börja med 07");
     return false;
   }
   if (c.value.search(/^07[0-9]{8}$/)==-1) {
     c.focus();
     alert("Ditt mobilnr måste vara exakt " +
           "10 siffror, utan bindestreck!");
     return false;
   }

   else

   {

   document.write("Det var ett korrekt formaterat mobiltelefonnummer");
   return true;

   }

} </script>
```

Länkar

- Länkar

  Reguljära uttryck finns på nätet, t.ex
  - http://www.gt.kth.se/PageMaster98/js/rexp.html
  - Mycket fler sidor kan finnas via www.google.com söker efter "javascript reguljära uttryck "

Lite mer om händelsehantering

Händelsehantering gör som tidigare nämnt, att JavaScriptprogrammen kan styras av händelser i webbläsar-fönstret.

- onClick/onDblClick (när användaren klickar eller dubbel klickar på något)
- onBlur (när ett fönster eller formulärobjekt mister fokus)
- onLoad/onUnLoad (Inträffar efter det att något laddats eller stängs)
- onMouseOver/onMouseOut (När pekaren flyttas över resp. från en länk.
- onKeyDown/onKeyPress/onKeyUp (När en tangent nedtrycks/Nedtrycks och släpps/släpps)

- Exempel på lite händelser finns i handelser.htm på kurssidan.

Händelsehanterare	Vad den påverkar	När det händer
onAbort	bilder	När bildens inladdning avbryts
onBlur	fönster, frames, formulärobjekt	När t.ex pekaren lämnar en textbox. Fokus lämnar objektet. Undantag är hidden.
onChange	Input, select och textområden.	När en använd. byter värdet på ett element och det förlorar inmatnings fokus. Används för formulär kontroll.
onClick	Sök på Internet eller i litteratur. för	
onDblClick	mer info om händelsehanterare.	
onDragDrop		
onError		
onFocus		
onKeyDown		
onKeyPress		
onKeyUp		
onLoad		
onMouseOut		
onMouseOver		
onMove		
onReset		
onResize		
onSelect		
onSubmit		
onUnload		

HÄNDELSEKATEGORIER

objekt	händelsetyp	tag
window	fönsterhantering	BODY, FRAMESET
	laddning av dokument	
form	komponent-hlser	INPUT*, TEXTAREA
anchor	länk-hlser	A
Image	bildhantering	IMG

HÄNDELSEHANTERARE

hanterare	objekt/TAG
onBlur	window (BODY), Text, Select, Textarea, Password (INPUT)
onChange	Text, Select, Textarea, FileUpload (INPUT)
onClick	Anchor (A), Button (INPUT)
onError	Image (IMG), window (måste skrivas)
onFocus	window (BODY), Text (INPUT)
onKeyDown	document, Image, Link, Textarea
onKeyUp	
onLoad	Image (IMG), window (BODY, FRAMESET)
onMouseDown	document, Link, Button
onMouseOut	Area, Link
onMouseOver	Link
onReset	Form
onSubmit	Form
onUnload	window

* INPUT har attributet TYPE som beskriver formulärelementet

Händelser för länkar, Area och Link

- onClick, onMouseOver, onMouseOut
- Klickbara bilder med hot spots kan sättas som
- Bildväxlare (onMouseOver) Ofta använd på webbplatser.

Image-objektets händelser

- onLoad
- onAbort
- onError Exempel (se bifogad fil på kursplatsen)
- Exempel på en animation gjord med Javascript. (se bifogad fil på kursplatsen)