

JavaScript + DOM

Lorem ipsum dolor sit amet, consectetur adipiscing elit

Dagens föreläsning

- Bakgrund
- Våra verktyg
- Grunderna i JavaScript
- DOM
- Händelser i JavaScript

Bakgrund

Vad är Javascript och varför ska vi bry oss?

Språket Javascript

- Ursprungligen avsett för att göra webbsidor roligare
 - Bra stöd för att manipulera DOM
 - Bra stöd i de flesta webbläsare
- Körs numera lite över allt
- Har numera ett bra stöd av tredjepartsbibliotek

Språket Javascript

- Syntaxen liknar C, Java, etc
- Löst typat – inga explicita variabeltyper
- Multiparadigm:
 - Händelsedrivet
 - Funktionellt
 - Prototypbaserat (ungefär som objektorienterat)
 - Imperativt

Exempel på användning

- Dynamiska webbsidor
- Serverapplikationer med Node.JS
- Mobilapplikationer med Cordova
- Spelutveckling med Unity
- Robotstyrning med Nodebots

Lite historik



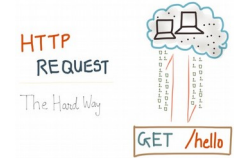
1995: Netscape



1996: JScript



1997: ECMAScript



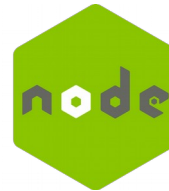
2005: AJAX



2006: JQuery



2008: V8-motorn



2010: Node.JS



2015: ES 6

Våra verktyg

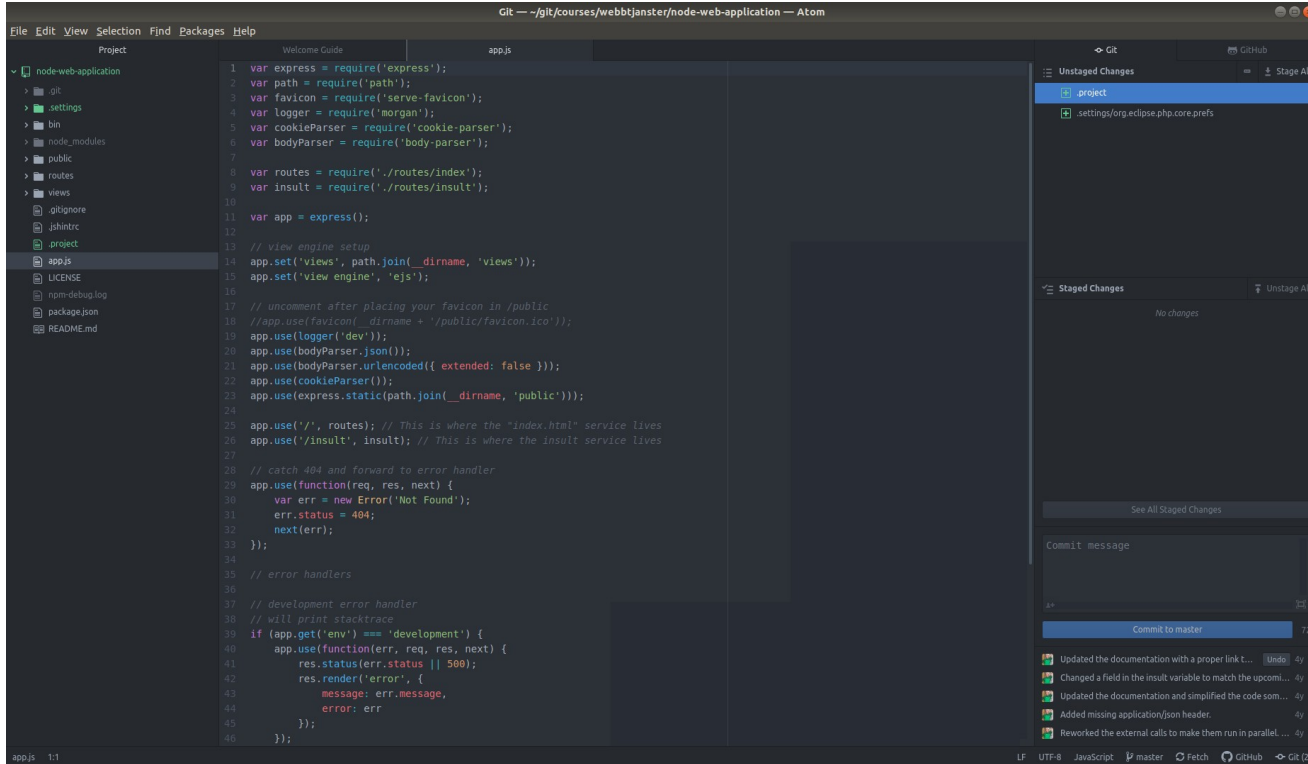
Våra verktyg

Textredigerare: Atom

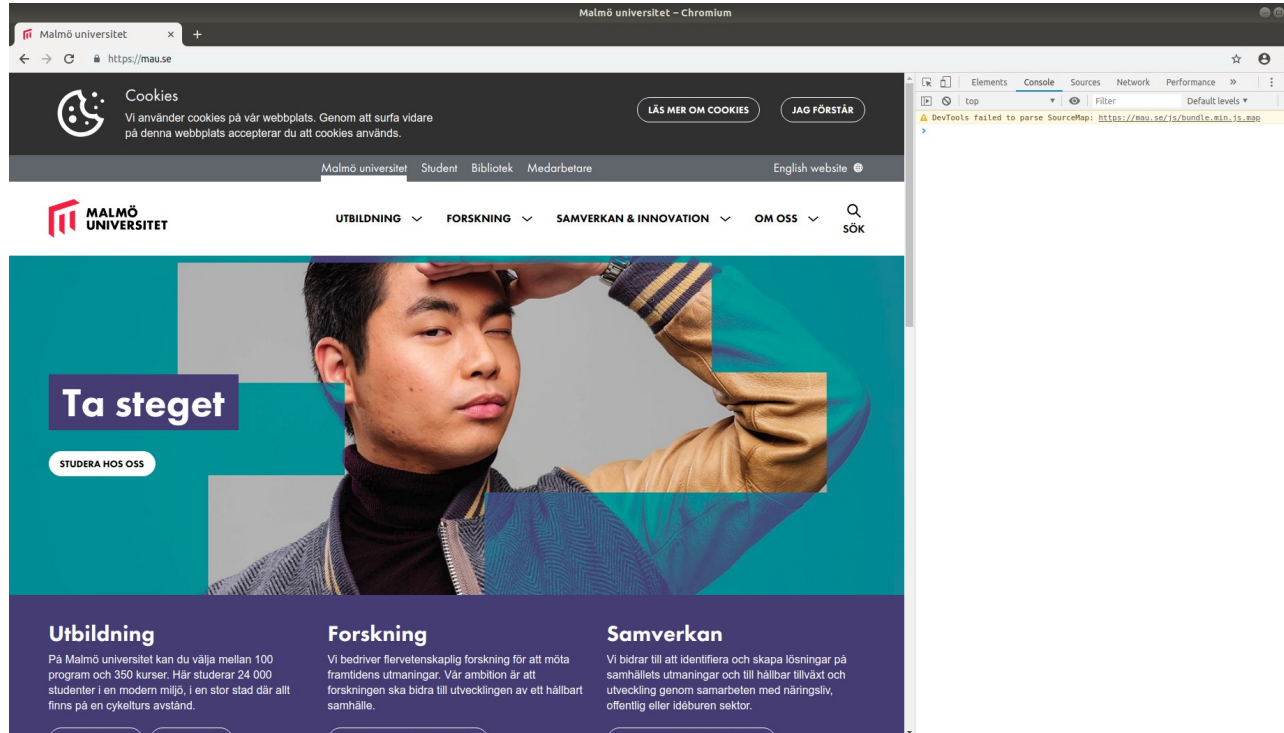
Webbläsare: Chromium med Chrome Developer Tools

Onlineverktyg: JSFiddle

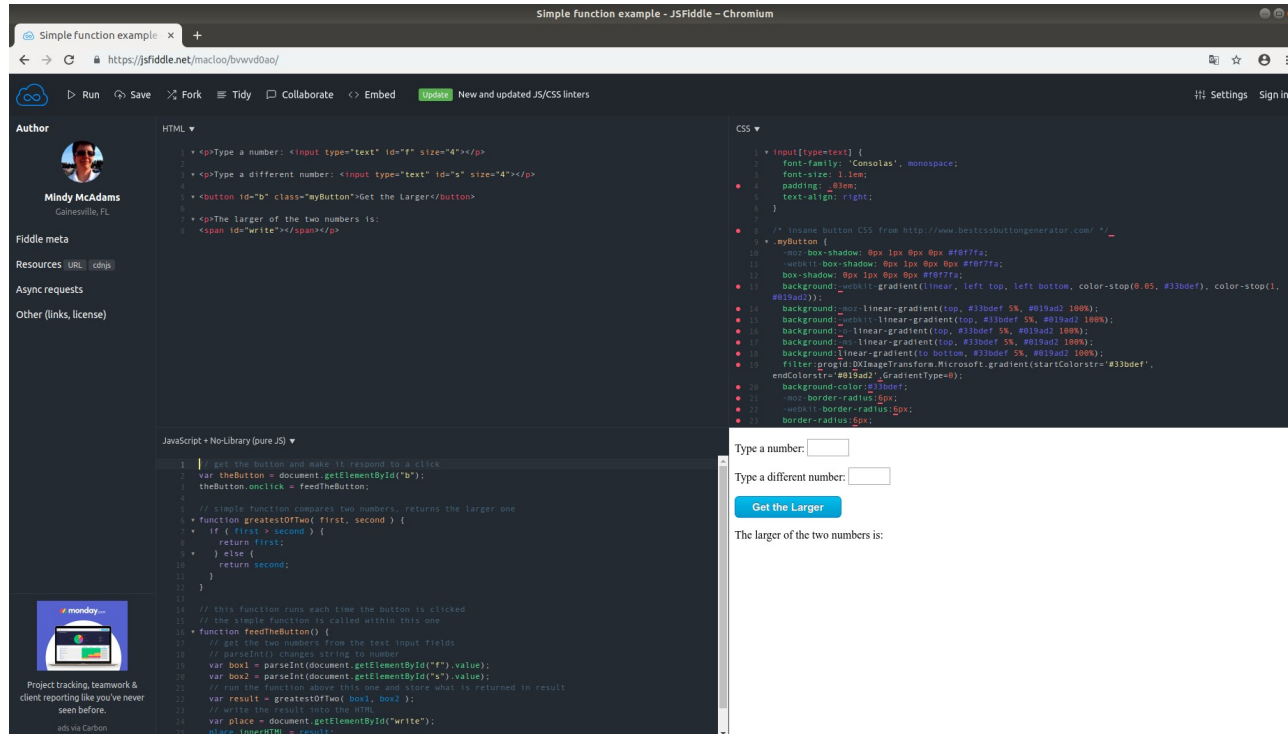
Textredigeraren Atom



Webbläsaren Chromium



Onlineverktøget JSFiddle



Grunderna i JavaScript

Satser

Instruktioner i Javascript kallas *statements* eller *satser*.

Varje sats *kan* avslutas med ett semikolon (eller radbrytning)

```
console.log("hello, world!");
```

Flera satser kan placeras i ett block. Dessa avgränsas med *måsvingar* eller *kryllparenteser*

```
{  
  console.log("hello");  
  console.log("world");  
}
```

Tilldelning

Variabler får sina värden med hjälp av ett = -tecken

```
a = 1; // Tilldelning  
b = true; // Tilldelning
```

If- och switch-satser

If-satser används för att hantera villkor:

```
if (a == 2) {  
    // Do something  
} else if (a == 4) {  
    // Do something  
} else {  
    // Do something else  
}
```

Switch-satser fungerar på ungefär samma sätt:

```
switch (a) {  
    case 2:  
        // Do something  
        break;  
    case 4:  
        // Do something  
        break;  
    default:  
        // Do something else  
}
```


Loopar

Två typer av loopar: while och for

```
while (true) {  
    // Do something  
}
```

```
for (var i = 0; i < 10; i++) {  
    // Do something  
}
```

Variabler

- Löst typade (sen nästa slide)
- Deklareras på tre sätt:
 - var – äldre sätt, synligt för “alla”
 - let – nyare sätt, synligt endast i aktuellt block
 - const – nyare sätt, konstant och synligt i aktuellt block

Operationer

// Aritmetik

```
a = 1 + 2; // Addition
a = 2 - 1; // Subtraktion
a = 2 * 2; // Multiplikation
a = 2 / 2; // Division
a++; // a = a + 1
a--; // a = a - 1
b = !b; // B är nu falskt
```

// Jämförelser

```
a == 2; // Är a lika med två?
a != 2; // Är a skilt från två?
a < 2; // Är a mindre än två?
a > 2; // Är a större än två?
a && b; // Är a och b båda sanna?
a && !b; // Är a sant och b falskt?
a || b; // Är a eller b sanna?
```

Variabler

```
var a; // Initialiserad, inget värde
var b = 1; // Initialiserad, har ett värde
let c = true; // Initialiserad, har ett värde
const d = "hello"; // Initialiserad, har ett värde
console.log(a); // Skriver ut "undefined"
console.log(b); // Skriver ut "1"
console.log(c); // Skriver ut "true"
console.log(d); // Skriver ut "hello"

b = 5; // Nytt värde!
console.log(b); // Skriver ut "5"

d = "goodbye"; // Här går det fel!
```

Datatyper

Javascript har fem datatyper (ish...)

- String
- Number
- Boolean
- Object
- Array

Strängar - String

En lista av tecken:

```
var a = "hello, world!";
```

Längden av en sträng:

```
console.log(a.length); // Skriver ut "13"
```

Hämta ut ett tecken ur en sträng:

```
console.log(a[2]); // Skriver ut "l"
```

Tal - Number

Vanliga hel- och flyttal:

```
var a = 1; // Heltal
```

```
var b = 1.5; // Flyttal
```

Och en specialare: NaN!

```
var c = NaN; // Inte ett nummer, men ändå ett nummer.
```

Sant/falskt - Boolean

Representerar värdena sant och falskt:

```
var iAmWeazel = true;
```

```
var iRBaboon = false;
```


Objekt - Object

Används för att hålla koll på många värden. Kan innehålla *vad som helst!*

```
var teacher = {  
  name: "Johan",  
  salary: 10000,  
  colleague: {  
    name: "Anton",  
    younger: true  
  }  
};
```

Vektorer/listor - Array

En typ av objekt som håller koll på värden i en specifik ordning:

```
var a = new Array();  
a[0] = 1;  
a[1] = "two";  
a.push(3);  
console.log(a); // Skriver ut "[1, "two", 3]"  
  
var b = [1, "two", 3]; // Likadan som a!  
console.log(b); // Skriver ut "[1, "two", 3]"
```

Datatyper

Några bonustyper

- null
- undefined
- function

Funktioner

Funktioner är återanvändbara samlingar av satser, dvs en eller flera rader av kod. De kan ta in *argument* och ge ett *returvärde*.

```
function a(a, b) {  
  return a + b;  
}
```

Funktioner som variabler

Funktioner kan även vara variabelvärden:

```
var anka = function (a, b) {  
  return a + b;  
}  
console.log(anka(1, 2)); // Skriver ut "3"
```

Det gör att de kan anropas lite hur som helst:

```
var b = {  
  name: "addition"  
};  
b["add"] = anka;  
console.log(b.add(1, 2)); // Skriver ut "3"
```

DOM

Vad är DOM?

HTML DOM (Document Object Model):

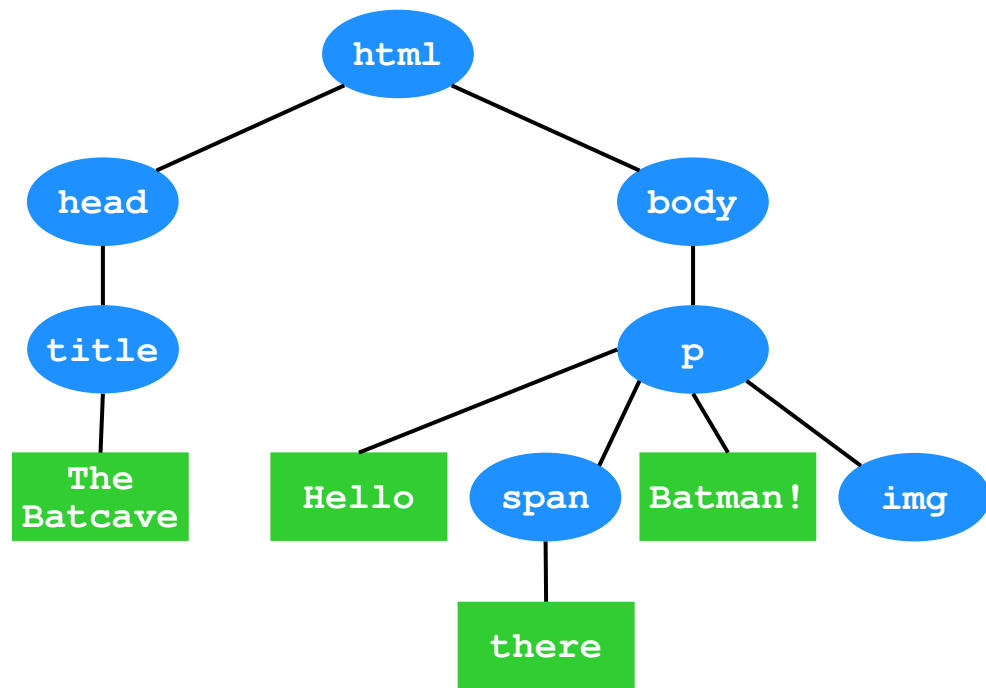
- En representation av HTML-dokumentet som ett träd av element – noder.
- HTML-objektet kan ses som ett stort objekt

Vad är DOM?

```
<!DOCTYPE html>
<html>
<head>
  <title>The Batcave</title>
</head>

<body>
  <p>
    Hello <span>there</span> Batman!
    
  </p>
</body>

</html>
```



Kom åt element/noder

Det finns fyra sätt att komma åt element i DOM-trädet:

- `document.getElementById(id)` → ett element
- `document.getElementsByTagName(tagName)` → en samling av element
- `document.getElementsByClassName(className)` → en samling av element
- `document.querySelector(query)` → ett element

Hitta alla element av en specifik typ

```
var elements = document.getElementsByTagName("p");
```

Hittar alla p-element och returnerar ett objekt av typen HTMLCollection, som liknar ett array-objekt:

```
var el = elements[0]; // Första elementet i samlingen
```

```
var size = elements.length; // Antal element
```

ID kontra klass i HTML

- Enskilda HTML-element (kan) identifieras med ett *ID*:
 - Ett ID består av en sträng
 - ID:n är unika och kan bara finnas en gång per dokument
 - Refereras till som `#mitt_id`
- Ett element kan tillhöra en *klass*:
 - Ett klassnamn består av en sträng
 - Flera element kan tillhöra samma klass.
 - Refereras till som `.min_klass`

Exempel på användande av ID

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
  <title>Sample Site</title>
```

```
</head>
```

```
<body>
```

```
  <p>
```

```
    Hello <span id="location">there</span> Batman!
```

```
    
```

```
  </p>
```

```
  <button onclick="relocate()">Change text</button>
```

```
<script>
```

```
  function relocate() {
```

```
    document.getElementById("location").innerHTML = "here";
```

```
  }
```

```
</script>
```

```
</body>
```

```
</html>
```

Vårt ID

Vårt ID

Exempel på användande av klass

```
<!DOCTYPE html>
<html>
<head>
  <title>Sample Site</title>
  <style>
    li.real_deal {
      color: red;
    }
  </style>
</head>

<body>
  <h1>Batman's gadgets</h1>

  <ul>
    <li class="real_deal">Batarang</li>
    <li class="real_deal">Batmobile</li>
    <li class="bad_pun">Batteries</li>
  </ul>
</body>

</html>
```

Den här regeln
matchar de här elementen

Exempel på användande av båda

```
<!DOCTYPE html>
<html>
<head>
  <title>Sample Site</title>
</head>

<body>
  <h1>Batman's gadgets</h1>

  <ul id="gadget_list">
    <li class="real_deal">Batarang</li>
    <li class="real_deal">Batmobile</li>
    <li class="bad_pun">Batteries</li>
  </ul>
</body>

</html>
```

Query Selector

```
var element = document.querySelector(".someClass");
```

Returnerar den första noden av klassen `someClass`.

Kan användas med:

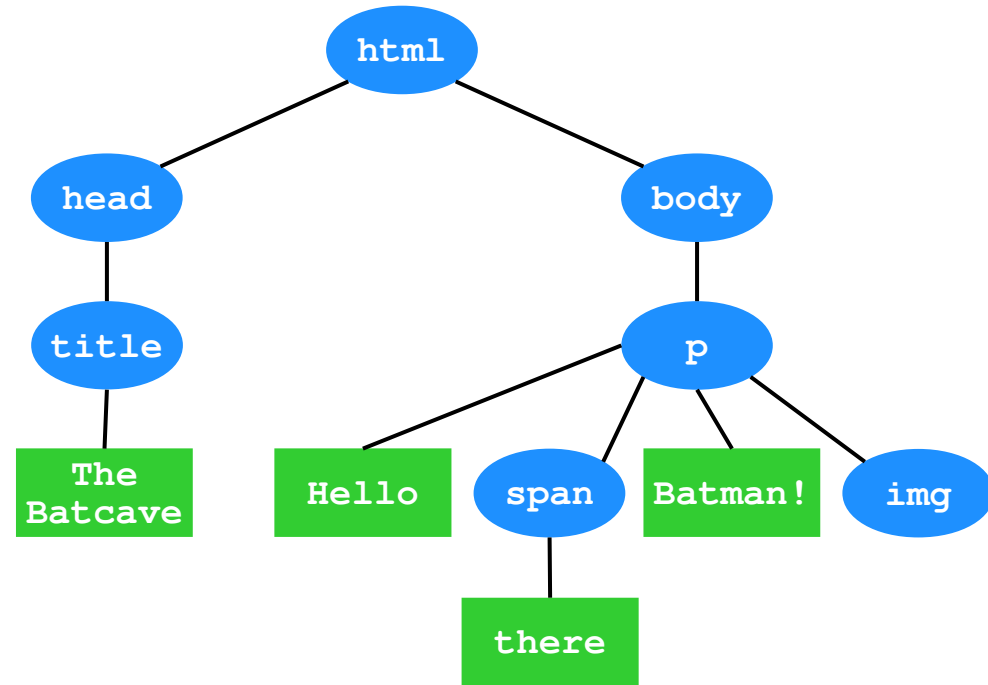
- Taggnamn: `document.querySelector("tag");`
- Klassnamn: `document.querySelector(".someClass");`
- ID: `document.querySelector("#someID");`

Navigera i DOM

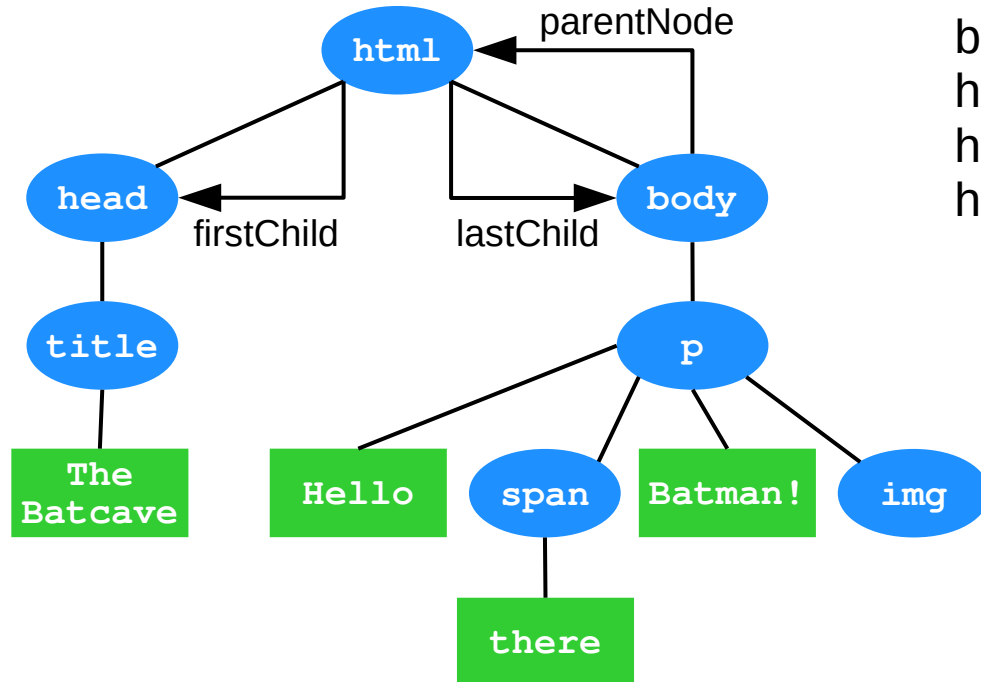
```
<!DOCTYPE html>
<html>
<head>
  <title>The Batcave</title>
</head>

<body>
  <p>
    Hello <span>there</span> Batman!
    
  </p>
</body>

</html>
```

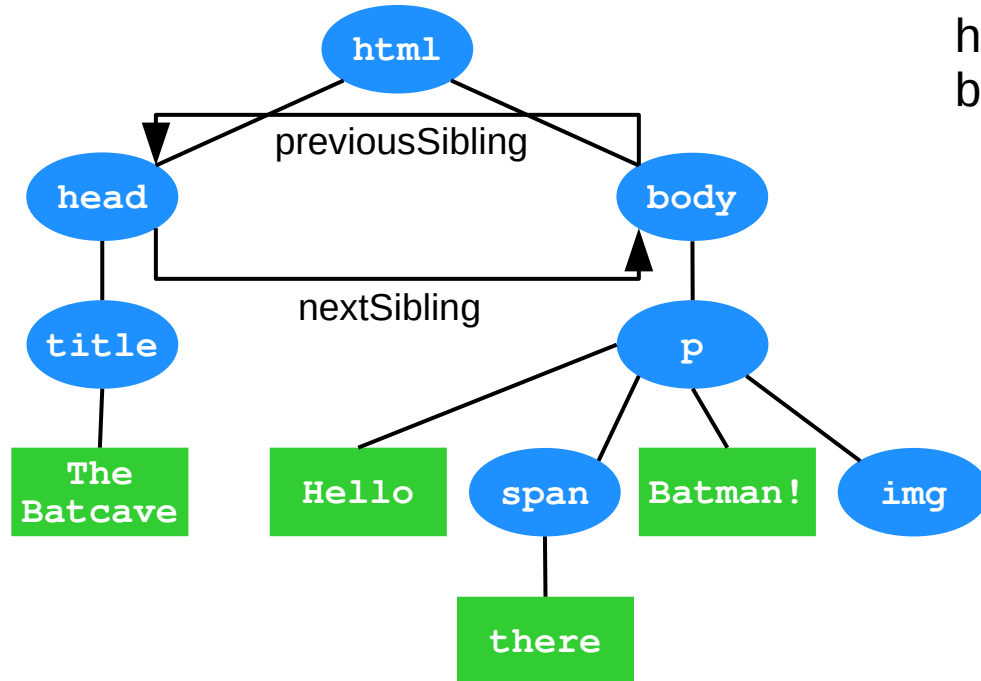


Navigera i DOM – föräldrar och barn



```
body.parentNode → html  
html.firstChild → head  
html.lastChild → body  
html.childNodes[1] → body
```

Navigera i DOM – syskon



`head.nextSibling` → `body`
`body.previousSibling` → `head`

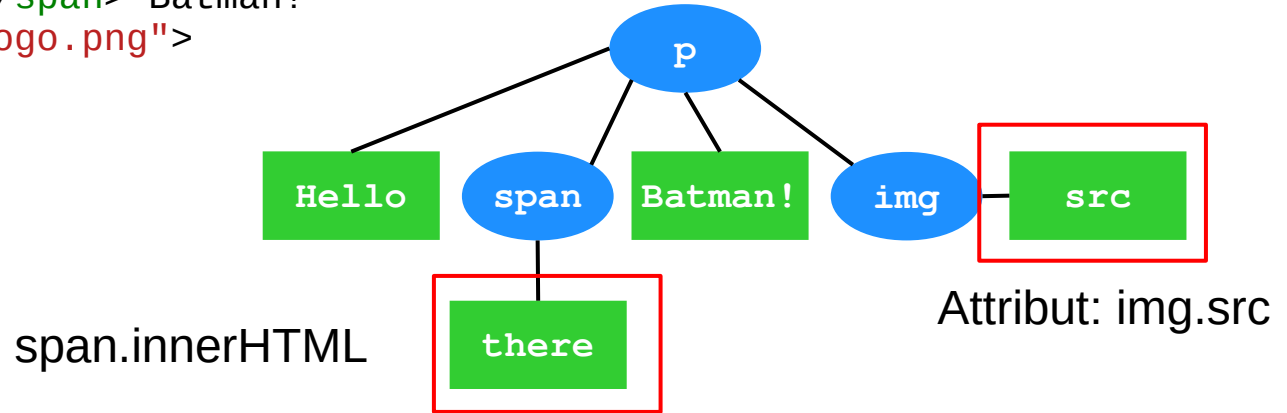
Modifiera en nod

Vi behöver känna till två typer av egenskaper hos en nod:

- InnerHTML
 - Den text som “innesluts” av ett HTML-element
 - *Kan* vara HTML-kod
- Attribut
 - Förändrar ett elements beteende. Exempel:
 - src
 - style

Vad är vad?

```
...  
<p>  
  Hello <span>there</span> Batman!  
    
</p>  
...
```



Arbeta med innerHTML

```
<!DOCTYPE html>
<html>
<head>
  <title>Sample Site</title>
</head>

<body>
  <h1>Welcome to the Batcave!</h1>

  <p>
    Alfred is <span id="status">out</span>.
  </p>

  <script>
    var status = document.getElementById("status");
    status.innerHTML = "in";
  </script>
</body>

</html>
```

Arbeta med attribut

```
<!DOCTYPE html>
<html>
<head>
  <title>Sample Site</title>
</head>

<body>
  <h1>Welcome to Arkham Asylum!</h1>

  <h2>Inmate of the month:</h2>
  

  <script>
    var inmate = document.getElementById("inmate");
    inmate.setAttribute("joker.jpg");
  </script>
</body>

</html>
```

Special: attribut-noder och CSS

- Attributet `style` (inline CSS) är en attribut-nod och hanteras så här:

```
...  
<body>  
  <h1>Welcome to Wayne Mansion!</h1>  
  
  <p>  
    Alfred is <span id="status" style="color: red">out</span>.  
  </p>  
  
  <script>  
    var status = document.getElementById("status");  
    status.innerHTML = "in";  
    status.style.color = "green";  
  </script>  
</body>  
...
```

Lägg till nya element till trädet

Vi skapar nya element genom att använda oss av

```
document.createElement("tag_name")
```

Nya element behöver ofta text. Det skapas så här:

```
document.createTextNode("My text")
```

Dessa sammanfogas sedan:

```
var p = document.createElement("p");  
var textNode = document.createTextNode("Batman");  
p.appendChild(textNode);
```


Lägg in nya element till trädet

Vi har två sätt att lägga till nya element till trädet

- `element.appendChild(node)`

lägg node sist bland barnen till element

- `element.insertBefore(node, child)`

lägg till före elementet child

Exempel på appendChild()

```
<!DOCTYPE html>
<html>
...

<body>
  <h1>Batman's gadgets</h1>

  <ul id="gadget_list">
    <li id="batarang" class="real_deal">Batarang</li>
    <li id="batmobile" class="real_deal">Batmobile</li>
    <li id="batteries" class="bad_pun">Batteries</li>
  </ul>

  <script>
    var batcave = document.createElement("li");
    var label = document.createTextNode("Bat cave");
    batcave.appendChild(label);

    var parent = document.getElementById("gadget_list");
    parent.appendChild(batcave);
  </script>
</body>

</html>
```

Exempel på insertBefore()

```
<!DOCTYPE html>
<html>
...

<body>
  <h1>Batman's gadgets</h1>

  <ul id="gadget_list">
    <li id="batarang" class="real_deal">Batarang</li>
    <li id="batmobile" class="real_deal">Batmobile</li>
    <li id="batteries" class="bad_pun">Batteries</li>
  </ul>

  <script>
    var batcave = document.createElement("li");
    var label = document.createTextNode("Bat cave");
    batcave.appendChild(label);

    var parent = document.getElementById("gadget_list");
    var batteries = document.getElementById("batteries");
    parent.insertBefore(batcave, batteries);
  </script>
</body>

</html>
```

Ta bort element från trädet

Vi har två sätt att ta bort element från DOM-trädet:

- `parent.removeChild(child)`
tar bort ett element
- `parent.replaceChild(element1, element2)`
ersätter ett element

Ta bort ett element

```
<!DOCTYPE html>
<html>
...
<body>
  <h1>Batman's gadgets</h1>

  <ul id="gadget_list">
    <li id="batarang" class="real_deal">Batarang</li>
    <li id="batmobile" class="real_deal">Batmobile</li>
    <li id="batteries" class="bad_pun">Batteries</li>
  </ul>

  <script>
    var parent = document.getElementById("gadget_list");
    var child = document.getElementById("batmobile");
    parent.removeChild(child);
  </script>
</body>

</html>
```

Ta bort ett element

```
<!DOCTYPE html>
<html>
...
<body>
  <h1>Batman's gadgets</h1>

  <ul id="gadget_list">
    <li id="batarang" class="real_deal">Batarang</li>
    <li id="batmobile" class="real_deal">Batmobile</li>
    <li id="batteries" class="bad_pun">Batteries</li>
  </ul>

  <script>
    var child = document.getElementById("batmobile");
    child.parentNode.removeChild(child);
  </script>
</body>

</html>
```

Ersätt ett element

```
<!DOCTYPE html>
<html>
...

<body>
  <h1>Batman's gadgets</h1>

  <ul id="gadget_list">
    <li id="batarang" class="real_deal">Batarang</li>
    <li id="batmobile" class="real_deal">Batmobile</li>
    <li id="batteries" class="bad_pun">Batteries</li>
  </ul>

  <script>
    var batcave = document.createElement("li");
    var label = document.createTextNode("Bat cave");
    batcave.appendChild(label);

    var parent = document.getElementById("gadget_list");
    var batteries = document.getElementById("batteries");
    parent.replaceChild(batcave, batteries);
  </script>
</body>

</html>
```

Händelser i JavaScript

Vad är händelser?

En händelse är ett meddelande som skickas till JavaScript-motorn, och som sedan påverkar körningen av programmet.

Exempel:

- Användaren klickar på en knapp
- Ett AJAX-anrop (dvs HTTP) slutförs
- En timeout tar slut
- Ett formulär skickas

Vi kan lyssna efter dessa händelser och reagera på dem!

Vanliga händelser i JavaScript/DOM

- Blur – när ett objekt förlorar fokus
- Click – när användaren klickar på ett objekt
- Focus – när ett objekt får fokus
- Load – när objektet är färdigladdat
- Mouseover – när muspekaren svävar över objektet
- Select – när ett val i en drop down-menyn väljs
- Submit – när ett formulär skickas

Reagera på händelser

Körning av JavaScript i webbläsaren är *asynkron*. JS-motorn har en lista av kod som den kan köra.

Motorn kör kod så länge den kan, en funktion i taget, och delegerar allt hårt arbete till webbläsaren.

När webbläsaren är färdig med ett uppdrag (eller får en input från användaren) läggs ett uppdrag i JavaScript-motorns att göra-lista.

Metoden `.addEventListener()`

```
element.addEventListener(event, function, useCapture);
```

- Parametern `event` är en sträng som anger vilken händelse vi vill lyssna efter.
- Parametern `function` är en funktionspekare och berättar vilken funktion som ska köras när händelsen inträffar. Den kan vara anonym.
- Parametern `useCapture` berättar om vi vill använda event bubbling eller event capturing. Överkurs, och helt frivilligt att använda.
- Ett element kan ha flera event listeners, även per händelse!

Exempel med addEventListener()

```
<!DOCTYPE html>
<html>
...

<body>
  <h1>Welcome to the Wayne Mansion intercom system!</h1>

  <span id="caller">Click to summon Alfred</span>

  <script>
    var caller = document.getElementById("caller");
    caller.addEventListener("click", callAlfred);

    function callAlfred() {
      alert("You called, Master Wayne");
    }
  </script>
</body>

</html>
```