TDP013 – Webbprogrammering och interaktivitet

Föreläsning 1:
Introduktion, HTML, CSS & JavaScript

Robin Keskisärkkä
Biträdande universitetslektor
Institutionen för Datavetenskap (IDA)

LINKÖPINGS UNIVERSITET

Kursens nyckelpersoner

- **Examinator:** Robin Keskisärkkä
- **Kursansvarig:** Robin Keskisärkkä
- **Labbassar:** Robin Keskisärkkä och Charlie Simonsson

- Kontakta Robin Keskisärkkä gällande allmänna frågor om kursen.
- Kontakta labbassar vid frågor som rör laborationer och projekt.

Kursens design

- I högskoleförordningen står det under mål för kandidatexamina bland annat att:

> *För kandidatexamen skall studenten - visa förmåga att söka, samla, värdera och kritiskt tolka relevant information i en problemställning samt att kritiskt diskutera företeelser, frågeställningar och situationer,*
>
> *- Högskoleförordningen* **(1993:100)**

- Kursen lägger stor vikt vid att ge studenterna möjlighet att själva söka efter den information som behövs för att lösa uppgifterna.
- Föreläsningarna är tänkta att introducera ämnena och laborationsuppgifterna för att komma igång.

Förändringar i kursupplägget

- Förtydligande av labbinstruktioner
- Tydligare deadlines för att feedback ska kunna ges snabbare
- Mer riktade föreläsningar för att underlätta och påskynda arbetet med laborationerna
- Fler kodexempel

- I läroplanen nämns PHP och SOAP men kommer att lägga fokus på framförallt JavaScript och AJAX.

Kursens innehåll

- Föreläsningar med fokus på olika webbtekniker
- Laborationerna behandlar HTML, CSS, JavaScript och MongoDB
- Reflektionsuppgift och seminarium kring etik inom mjukvarutveckling
- Programmeringsprojekt

Kursens tekniska innehåll

- Klientsida
  - HTML, CSS, JavaScript
- Serversida
  - Node.js
  - Express.js
  - MongoDB
- Verktyg för testning av kod på serversidan

Kursens examination

- Laborationer och projekt görs i första hand i par
- Laborationer och projekt kan i undantagsfall göras enskilt
- För godkänt i kursen krävs godkänt på samtliga laborationer. Graderat betyg (3–5) avgörs av projektet.
- Mer info finns på kurshemsidan.

Kursen ges på distans

- Kursen ges i första hand på distans men lokaler finns bokade för de som vill labba i sal.
- Föreläsningar, laborationshandledning och seminarier hålls via Teams enligt schemat.
- Viktigt att ni **registrerar er i Webreg** för att ni ska få tillgång till rätt kanaler i Teams
- Testa gärna att starta ett videosamtal i Teams och dela skärm med er labbpartner redan innan första laborationen
- För att koda tillsammans på distans kan ni ex. använda Visual Studio Code med Live Share-pluginet

Varför webbapplikationer?

- Inga installationer krävs
  - En version till alla operativsystem
  - Lägre tröskel för att “prova på”
- Kan köras på alla enheter med webbläsare
- Uppdateringar kan göras tillgängliga omedelbart utan att användaren behöver göra något
- Lägre underhållskostnader (alla användare kan ha samma version av webbappen)

HTML

Syntax och struktur

HTML5

```
<!doctype html>
<html>
   <head>
      <meta charset="utf-8">
      <title>Document 1</title>
   </head>
   <body>
      <p>The first and only paragraph.</p>
   </body>
</html>
```

- **Element**
  - start + slut
  - attribut + värden
  - innehåll
- Element kan nästlas men får inte överlappa

Syntax

- Whitespace (tab, mellanslag, nyrad) är inte signifikanta men krävs för att **kodstrukturen ska bli tydlig**
- Versaler och gemener spelar ingen roll men **använd små bokstäver för taggar**
- Taggarna bildar en trädstruktur

Tag-syntax

- Dokumenttyp

  <!doctype html>

- Självstängande tag

  <tag> (eller **<tag/>** eller **<tag />)**

- Start- och slut-tag

  <tag>content</tag>

- Tag med ett attribut

  <tag attribute> … **</tag>**

- Tag med ett attribut + värde

  <tag attribute=“value”> … **</tag>**

HTML definierar struktur

- **Semantisk användning**

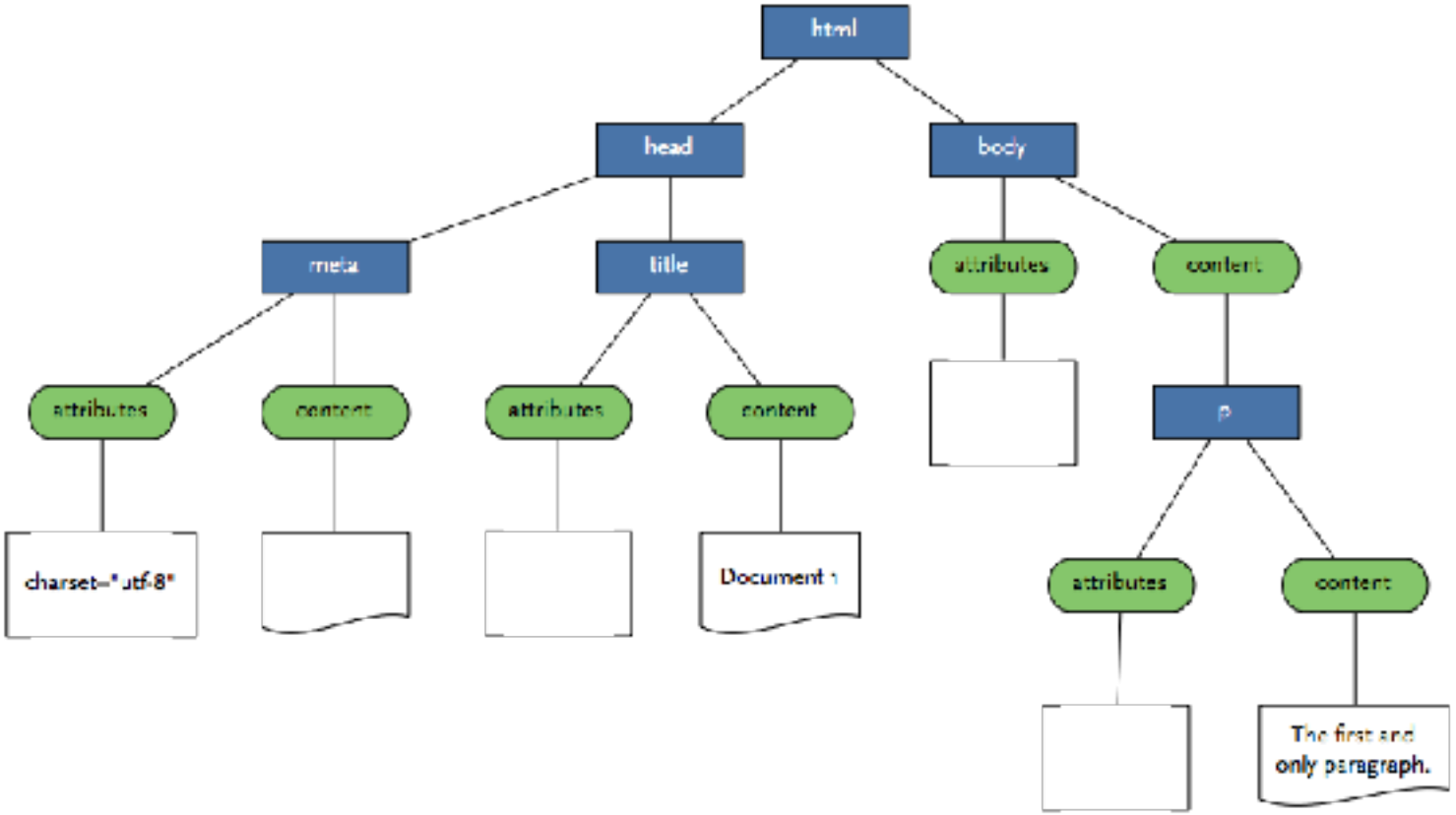
  för sökmotorer

- **Organiserar kod**

  förbereder för CSS styles

  Förbereder för interaktion

Document Object Model (DOM)



Kommentarer

- Använd kommentarer för att avaktivera HTML-kod eller kommentera kod

```
<!-- This is a comment -->
```

Rubriker

- Rubriker på olika nivåer

```
<h1>Heading 1</h1>
<h2>Heading 2</h2>
<h3>Heading 3</h3>
<h4>Heading 4</h4>
<h5>Heading 5</h5>
<h6>Heading 6</h6>
```

Paragrafer

Paragraph 1

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Paragraph 3

Paragraph 4

Länkar

```
<a href="url">Länktext</a>
```

Bilder

```
<img src=“./bigbang.jpg" alt="Bild föreställande big bang">
```

Strukturella element

- Två generiska element används för struktur
  - Block-element: ** **

    Används exempelvis för att organisera sidan i olika block
  - Inline-element: ** **

    Används exempelvis för att ändra utseende på delar i en text

Semantiska element i HTML5

- Semantiska element fungerar som div-taggar men kommunicerar något om syftet med taggen.

<header>

<nav>

<section>

<article>

<aside>

<figcaption>

<figure>

<footer>

http://www.w3.org/TR/html5/

Validering av HTML kod

- http://validator.w3.org
- Korrekt skriven HTML kod gör att webbläsaren inte behöver gissa hur den skall tolka koden
- Webbplatsen har större chans att se ut som du tänkt dig i alla webbläsare
- I laborationsserien ska koden valideras som HTML5
- Om du har fel så rätta ett i taget tills du får ”grönt”
  - Ibland klagar validerare på märkliga saker. Om ni känner er övertygade om att ni har rätt och valideraren fel så är det okej.

CSS

Syntax och struktur

CSS (Cascading Style Sheets)

- CSS specificerar
    - visuell stil
    - layout
- CSS specificerar inte struktur!
- HTML-element ärver CSS-properties från deras föräldrar

Stilaspekter

- Färger och kanter

  Ex. background color, borders runt element

- Typografi

  Ex. teckensnitt, textstorlek, höjd på rader

- Storlek på element

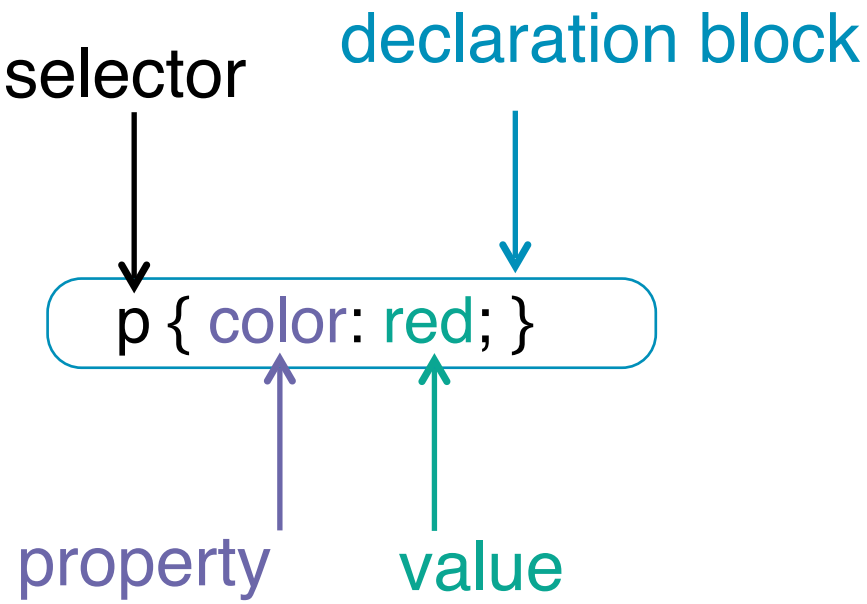
Layout-aspekter

- Interaktion mellan element
    - Vad händer när två element försöker placeras på samma ställe?
    - Vad händer om alla element inte får plats?
- Storlek på element
- Positionering av element

CSS-stylesheet

- En textfil med filändelsen **.css**
- Innehåller en samling style-specifikationer
- Ett HTML-dokument kan länka till **ett eller flera stylesheets**
- Style-specifikationer utvärderas i ordning (uppifrån och ner) och efter specificitet
- Senare specifikationer kan köra över tidigare specifikationer
- … vissa undantag finns som användandet av !important

Syntax



Syntax

```
/* This is a comment */
body {
    font-family: Times, Serif;
    font-size: 16px;
    padding: 0px 0px 0px 0px;
}

h1 {
    font-family: Helvetica, Sans-Serif;
    font-size: 32px;
}
```

CSS-måttenheter

- https://developer.mozilla.org/en-US/docs/Web/CSS/length
- px: pixlar på skärmen
  - .. men en CSS-pixel är inte riktigt samma som en skärmpixel!
  - http://inamidst.com/stuff/notes/csspx
- em: ärvd font-size
  - 1em = ärvd font-size
  - 2em = dubbel ärvd font-size
- rem: som em men ärvt från rot-elementet
- pt: point (1pt = 1/72 inch)
- %: procent av ärvd storlek

HTML + CSS

- HTML-koden kan länka till CSS-filer
- CSS-kod kan skrivas direkt i HTML-kod men ska i labbarna skrivas i externa filer
- Länken från HTML till CSS görs med hjälp av <link> i header-taggen

page.html och style.css

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>A simple document</title>
    <link rel="stylesheet" href="./style.css">
  </head>
  <body>
    <h1>A heading</h1>
    <p>A paragraph</p>
  <body>
</html>
-
```

```
body {
    font-family: Times, Serif;
    font-size: 16px;
    padding: 0px 0px 0px 0px;
}

h1 {
    font-family: Helvetica, Sans-Serif;
    font-size: 32px;
}
```

Filvägar i HTML och CSS

- Filvägar relativa till nuvarande fil
- ./ pekar på på samma nivå i filstrukturen
- ../ pekar på en nivå upp i filstrukturen
- / syftar på hemsidans rot

  https://www-und.ida.liu.se/~yourliuid/index.html

  har roten

  https://www-und.ida.liu.se/
- Filvägar kan även vara absoluta

Exempel på CSS-properties

Några vanliga style-properties

- font-size: storlek på font
- font-family: namn på font
- line-height: höjd på textraden
- width: bredd på element
- height: höjd på element
- margin: avstånd till nästa element
- padding: avstånd från elementets kant till innehållet
- color: font-färg
- background-color: bakgrundsfärg på elementet
- border: border-style på elementet

Style-properties kan ta olika många argument

```
body {
    font-family: Times, Serif;
    padding: 0px 0px 0px 0px; /* samma som padding: 0px */
    border: 1px solid black;
}

h1 {
    font-family: Helvetica;
    padding: 10px 0px;
}
```

CSS-selectors

Selectors: välja element

- Välja en grupp av element

  “välj alla paragrafer och rubriker”

- Välja intilliggande syskon-element (samma nivå)

  “välj alla paragrafer som följs av en rubrik”

- Välj alla ättlingar (descendants)

  “välj alla bilder som ligger i en paragraf”

- Välj alla barn

  “välj alla listelement i listor inom en viss klass”

https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/Selectors

http://www.w3.org/TR/selectors/

Grupp av element

```
/* Target all h1, h2 and h3 element */

h1, h2, h3 {
    border: 2px solid #000;
}
```

Descendant-kombination

```
/* Select all li element that are nested within a nav element. */
nav li {
    color: #F00;
}
```

Child-kombination

```
/* Target all p elements that are children of a div */

div > p {
    border: 2px solid #000;
}
```

Intilliggande syskon

```
/* Target all p elements immediately followed by a h1 */

h1 + p {
   font-weight: bold;
}
```

Pseudo selectors

- :hover

  Välj elementet som musen hovrar över. Ex. ändra utsende på länk när musen är över den a:hover { … }

- :visited

  Välj element (dvs. länkar) som har besökts tidigare a:visited

CSS Reset

CSS-reset

- Browsers har egna definierade stylesheets som default
- Visa saker skiljer sig åt mellan browsers (ex textkensnitt, storlek på rubriker, osv.)
- **CSS-reset**: CSS-kod som har som syfte att nollställa alla properties till kända värden. Skapar en väldefinierad grund att bygga vidare på
- Populära varianter
  - **Eric Meyer's CSS reset från 2011**
    https://meyerweb.com/eric/tools/css/reset/reset.css
  - **Normalize.css**
    http://nicolasgallagher.com/about-normalize-css/
  - **Reboot, Resets, and Reasoning**
    https://css-tricks.com/reboot-resets-reasoning/

Klasser och ID:n

Vad är en klass (class)?

- Element kan associeras med **en eller flera** klasser
- Mer än ett element kan ha samma klass
- Klasser kan användas för styling av återkommande komponenter
- Prefixet . (punkt) används framför klassnamn i CSS-selectorer
- Exempel:
    - HTML

     eller med flera klasser

    - CSS

    .box, div.box, div.box.green-bg, green-bg

Exempel med flera klasser

```css
.wrapper {
  width: 960px;
}

.green-bg {
  background-color: green;
}

.yellow-bg {
  background-color: yellow;
}

.blue-fg {
  color: blue;
}

.box {
  border: 1px solid purple;
  width: 100px;
  height: 100px;
}
```

```html
<div class="wrapper">
  <div class="green-bg box”>
      Box 1
  </div>
  <div class="green-yellow box”>
      Box 2
  </div>
  <div class="box blue-fg”>
      Box 3
  </div>

  <!-- Order in the CSS matters -->
  <div class="green-bg yellow-bg box”>
      Box 4
  </div>
</div>
```

Vad är ett ID?

- Ett element kan ha **exakt ett ID**
- Alla ID:n måste vara unika i HTML-dokumentet
- ID:n kan användas för att lägga till en style till ett specifikt element
- Prefixet # används framför ID:n i CSS-selectors
- Exempel:
    - HTML

     eller <h1 id="main-heading">

    - CSS

    #footer, div#footer, h1#main-heading, #main-heading

Exempel i CSS

```
.infobox {
    font-family: Helvetica, Arial, Sans-Serif;
    font-size: 0.9em;
    background-color: #999;
    color: #000;
    border: 2px solid black;
}

#menu {
    background-color: #000;
    color: #FFF;
}
```

Färger och borders

Definiera färger med RGB

- RGB (red - green - blue) är en additiv färgmodell
- Värden från 0-255 (decimal) eller 0-F eller 00-FF (hex)
- Svart

  rgb(0, 0, 0) eller #000 eller #000000
- Vit

  rgb(255, 255, 255) eller #FFF eller #FFFFFF
- Lila

  rgb(128, 0, 255) eller #8000FF

Definiera färger: namngivna färger

- **CSS Level 1**: black, silver, gray, white, maroon, red, purple, fuchsia, green, lime, olive, yellow, navy, blue, teal, aqua
- **CSS Level 2 (revision 1)**: + orange
- **CSS Color Module Level 3**: + aliceblue, antiquewhite, aquamarine, azure, beige, bisque, blanchedalmond, blueviolet, brown, burlywood, cadetblue, chartreuse, chocolate, coral, cornflowerblue, cornsilk, …
- **CSS Color Module Level 4**: + rebeccapurple

background-color, color

```
h1 {
  color: #fff;
  background-color: #075488;
}
```

CSS borders

- Vissa typer av element kan ha en border (“kant”)
- En border har följande properties:

  width: linjens bredd

  style: linjens typ (exempelvis streckad)

  color: linjens färg
- Varje property kan definieras för varje sida (upp, höger, ner eller vänster)
- Hörnen kan även rundas med **border-radius**

CSS borders

- Shorthand-form (mindre kontroll)

  border: <width> <style> <color>

- Specifika properties

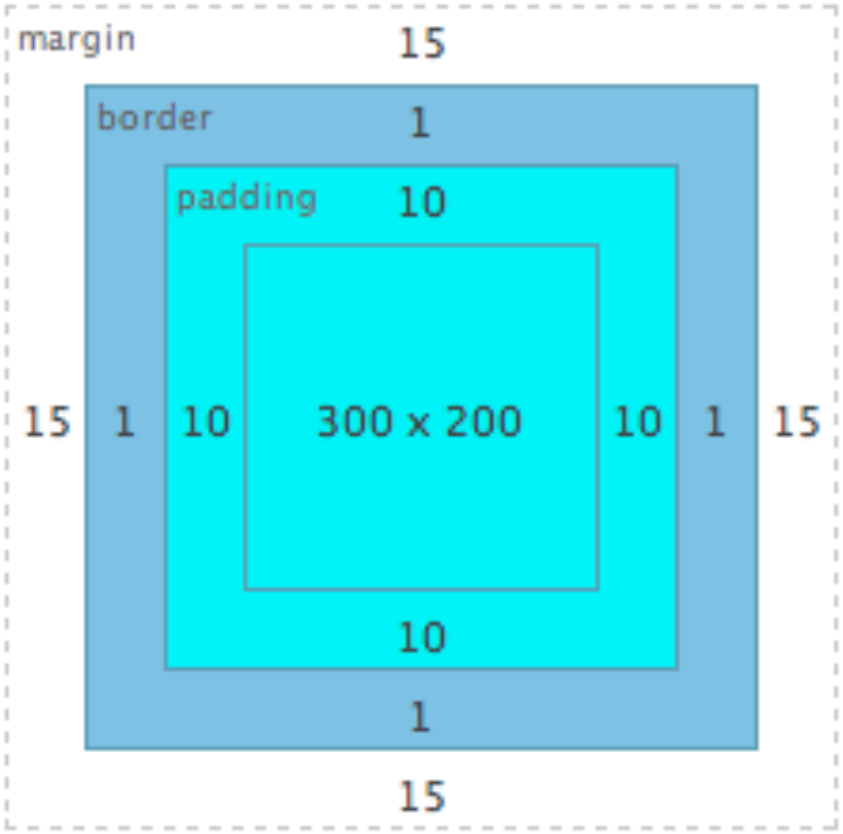
  border-style, border-width, border-color

  border-top, border-right, border-bottom, border-left

  border-top-style, border-top-color, ...

CSS: Boxmodellen & layout

Boxmodellen



Definiera padding

- padding: <north> <east> <south> <west>;
- padding-top: <value>;
- padding-right: <value>;
- padding-bottom: <value>;
- padding-left: <value>;

Definiera margin

- margin: <north> <east> <south> <west>
- margin-top: <value>;
- margin-right: <value>;
- margin-bottom: <value>;
- margin-left: <value>;

CSS: Layout-modeller

Layout-modeller i CSS

- **Normal flow + positioning**

  display: block | inline | inline-block

- **Flexbox**

  display: flex

- **Grid layout**

  display: grid

- **Floats**

  float: left | right

- **Table layout**

- **Multiple-column layout**

Vad borde man använda och lära sig?

- Varje element existerar inuti en layout-kontext
- Varje element kan ha en layout-kontext
- Bra att lär sig att hantera **normalt flow + positionering + floats** innan man lär sig **flex/grid** etc.
- Flexbox- och grid-layout mycket användbara men kan inte lösa allt

Normalt flow

- Element can bete sig på olika sätt
- Kontexterna for normalt flow är:

  inline

  block

  inline-block

- inline innebär att elementet ligger tillsammans med texten. Det generiska inline-elementet är
- block innebär att elementet ligger utanför texten. Det generiska block-elementet är . Ett block-element sträcker sig över hela bredden per default.
- inline-block är ett block som kan läggas tillsammans med text.

Display-property

```
/* The formatting context is set using the display property */

.infobox {
    display: block;
}

.question {
    display: inline;
}
```

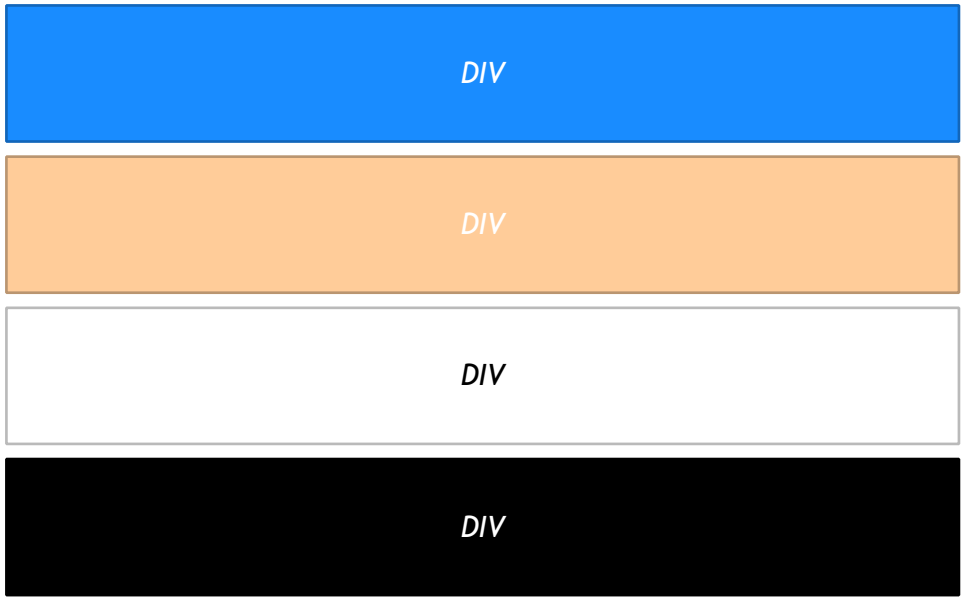
Fem positionsmodeller

- Position bestämmer hur ett block ska placeras ut på sidan i förhållande till andra
    - **static** (default): placerar ut element i den ordning de förekommer i dokumentet. Properties top, right, bottom, left och z-index har ingen effekt
    - **relative**: liknar static men tillåter att properties top, right, bottom, left och z-index används för att ändra positionen relativt till föregående element.
    - **absolute**: positionerar elementet i förhållande till närmsta icke-statiska förälder. Properties top, right, bottom, left och z-index används för att ändra positionen.
    - **fixed**: beter sig på samma sätt som absolute men i förhållande till webbläsarfönstret (viewport)
    - **sticky**: beter sig som relative tills dess att en specificerad offset uppfylls varefter den beter sig som fixed.
- Exempel: https://jsfiddle.net/4nepm12L/46/

Floats

- Syntax: float: left | right
- Vad händer? Float-element tas ur den normala kontexten och "flyter" inuti föräldrablocket.
- Elementen kan läggas placeras runt andra element automatiskt
- https://developer.mozilla.org/en-US/docs/Web/CSS/Containing_block

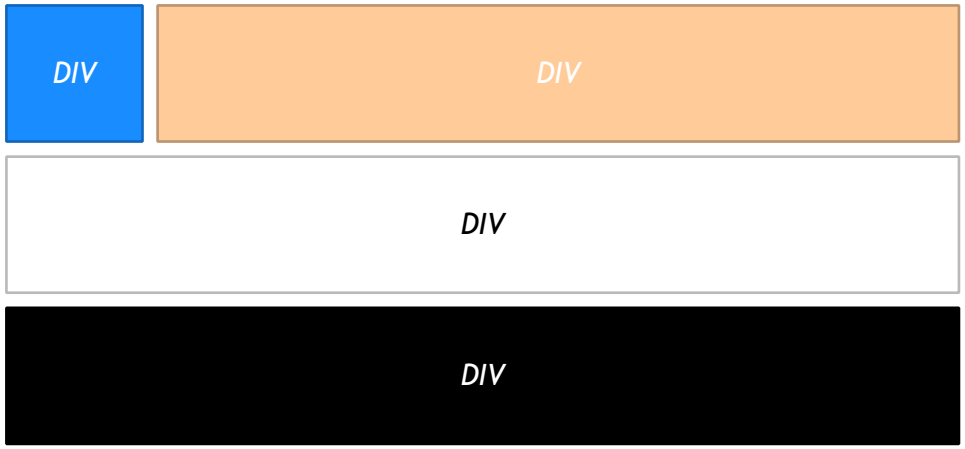
Floats: Exempel



Antag att vi har fyra div taggar efter varandra.

När webbläsaren ser nästa div tagg kommer den att göra en radbrytning och sedan rendera hela div-taggen.

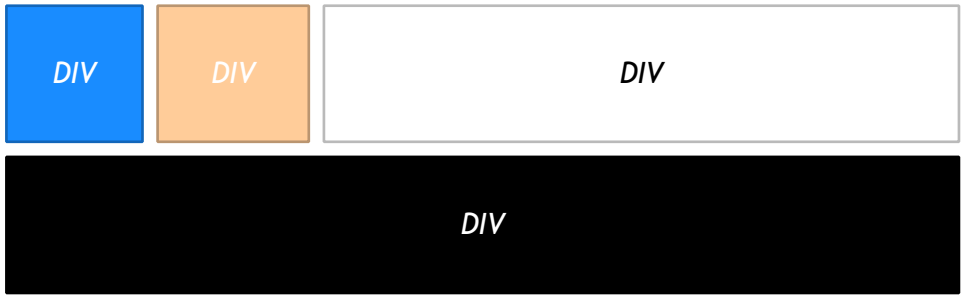
Floats: Exempel



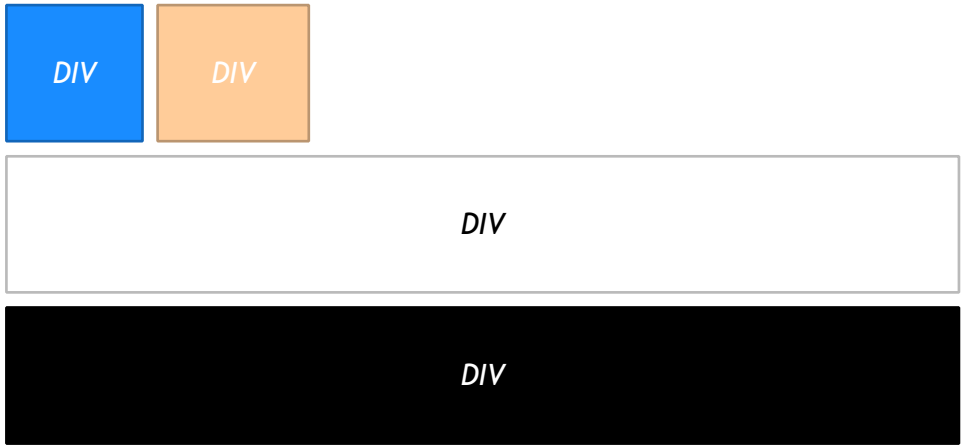
Om vi på den blå div-taggen sätter **float: left;** så placeras den så långt till vänster som möjligt och krymper till minsta tillåtna storlek. Efterkommande element hamnar sedan direkt efter.

div-taggen som är orange har inte float, så den vita hamnar på nästa rad.

Floats: Exempel



Om vi sätter **float: left;** på orange kommer den vita direkt efter.



Om vi vill undvika att det nästkommande elementet hamnar direkt efter den röda, så sätter vi **clear: left;** på det vita elementet.

"The clearfix hack"

- Används när man vill förhindra att ytterligare element flyter runt ett element

  clear: left | right | both

- Tolkning:
  *Ett element med **clear** kommer inte flyta upp så att det hamnar bredvid ett tidigare element. **Men** det förhindrar inte att efterkommande element flyter till elementet.*

- *Exempel:* https://jsfiddle.net/thdeg34p/29/

CSS-variabler

CSS-variables

- Definieras för en kontext (vanligtvist **:root**)
- Variabelnamn måste börja på med double-dash **(--)**
- Variabler kan bindas till giltiga CSS-värden
- Refereras till med hjälp av **var(varname)**

```
:root {
  --main-bg-color: brown;
}

.one {
  color: white;
  background-color: var(--main-bg-color);
  display: inline-block;
}
```

JavaScript: ES6

ECMAScript Edition 6: ES6

- JavaScript är en implementation av ECMAScript-specificationen
- Version 6 lanserades i juni 2015
- Mer eller mindre fullständigt stöd i alla moderna webbläsare
- Chrome, Firefox, Safari, Opera, Edge, Brave, ...
- Finns senare versioner men med mer begränsat stöd

Callback-funktioner

- JavaScript använder bara en tråd
- Funktioner som argument till funktioner
- Lämnar över ansvaret för att fånga upp data och event till den kallade funktionen
- Stor del av JavaScript och tredjeparts-bibliotek
- ”*När du har laddat ned bilden så gör det här.*”
- Vi kommer gå in närmare på callbacks och asynkrona anrop i nästa föreläsning

Tilldelningar och scope

- Olika sätt att deklarera variabler
    - **var** (ES5)
    - **let** (ES6)
    - **const** (ES6)
- Skillnaden ligger i scope
    - **var** har ett scope definierat av närmaste funktion
    - **let** och **const** har ett scope definierat av närmaste block
- Exempel: https://jsfiddle.net/08frseu1/43/

Viktigaste tillskotten i ES6

- arrow functions
- let + const
- iterators + for..of
- promises
- template strings
- classes
- modules

Arrow-functions

- Ett sätt att definiera anonyma funktioner
- Används mycket flitigt för att hantera ex. asynkrona anrop
- Exempel:

```
function hello(){
    console.log(“Hello world!”)
}
hello()

let hello = () => console.log(“Hello world!”)
hello()

// Callback-funktioner
do_time_consuming_thing.then(() => {
    console.log(“Hello world!”)
})
```

Tilldelningar och scope

```
var x = 1;
let y = 2;
const z = 3;
console.log(x, y, z); // outputs 1, 2, 3

z = 4; // Invalid assignment to const ‘z’!
```

Tilldelningar och scope

```
for(var x=0; x < 10; x++){
    console.log(x);
}
console.log(x); // 10, no problem

for(let y=0; y < 10; y++){
    console.log(y);
}
console.log(y); // ReferenceError
```

Tilldelningar och scope

```
function functionA() {
    let var1 = ‘Hello, world’;
    var var2 = ‘Hello, world’;
}

functionA();
var1; // ReferenceError
var2; // ReferenceError
```

Tilldelningar och scope

```
const v = [];
for(let i=0; i < 5; i++){
    v.push(() => i);
}

for(let x of v){
    console.log(x());
}

// 0, 1, 2, 3, 4
```

```
const v = [];
for(var i=0; i < 5; i++){
    v.push(() => i);
}

for(let x of v){
    console.log(x());
}

// 10, 10, 10, 10, 10
```

Objektorienterad programmering

- JavaScript använder prototyping till skillnad från Java/Python/C++
- JavaScript stödjer endast arv från en prototyp (ej multipelt arv)
- ”Vanlig” objektorienterad programmering i JavaScript är möjlig men kostar mer minne då varje objekt definierar sina egna funktioner

Klasser

- I ES5 är funktioner motsvarigheten till klasser där funktioner kan innehålla funktioner
- ES6 introducerar “syntactic sugar” för klasser vilket underlättar användandet rejält

Prototyping

```
function Animal() { // Constructor
    alert('Animal::Constructor');
}

Animal.prototype.talk = function() { // Method
    return "sound";
}

let myAnimal = new Animal();

Animal.prototype.whisper = function() {
    return "…";
}

myAnimal.scream = function() {
    return "SOUND!";
}

alert(myAnimal.whisper());

alert(myAnimal.talk());

alert(myAnimal.scream());
```

Arv

- Utökar egentligen bara ett objekt med fler prototyper
- Konstruktorn till subklassen måste refereras till explicit, annars används konstruktorn för föräldern
- Konstruktorn i föräldern måste anropas när arvet definieras

```
function Dog() {
    Animal.call(this);
    alert('Dog::Constructor');
}
Dog.prototype = new Animal();
Dog.prototype.constructor = Dog;
Dog.prototype.talk = function() {
    return "woff woff";
}

let myDog = new Dog();
alert(myDog.talk());
```

Syntax i ES6

- Syntax i ES6 är betydligt enklare!
- Exempel: http://jsfiddle.net/6g8pfyo0/16/

Cookies

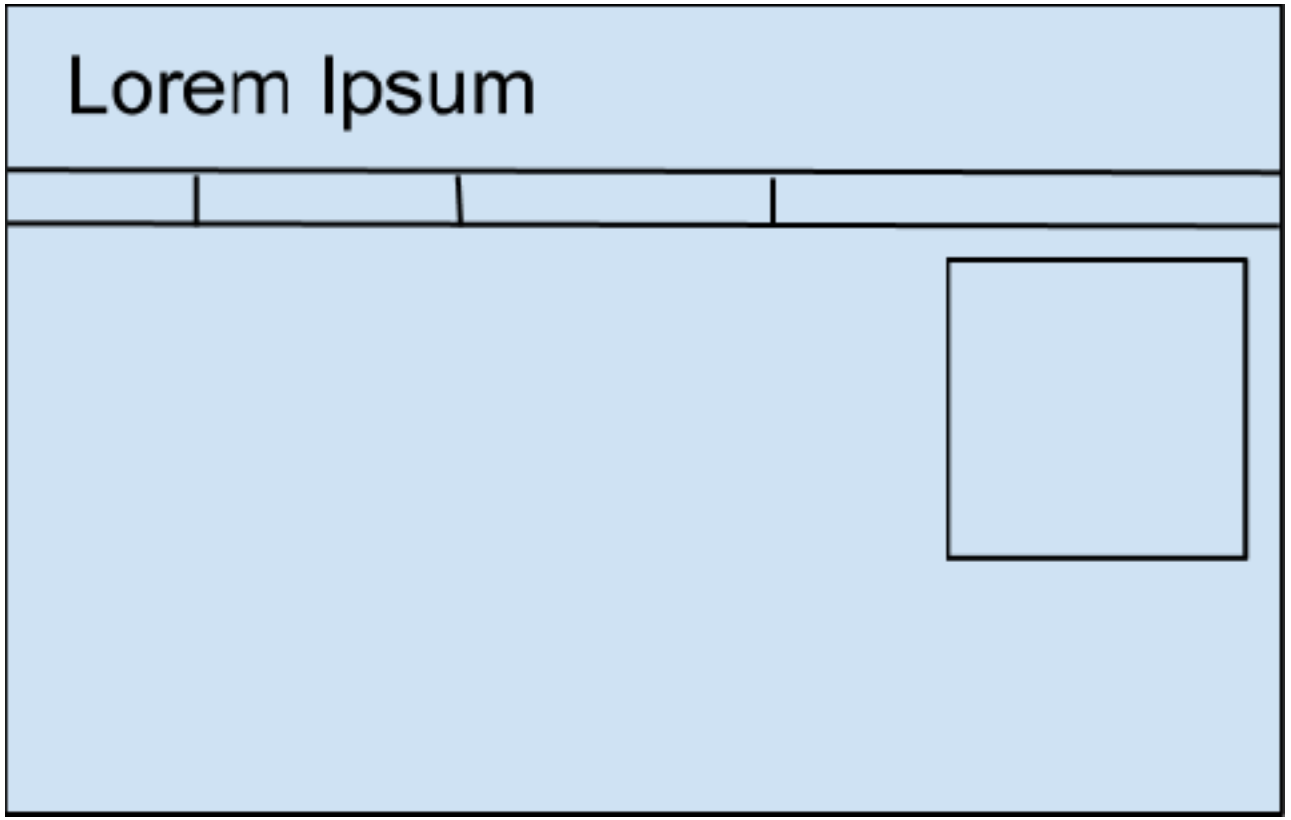
- Cookies är små mängder data som sparas i webbläsaren (max 4 kB)
- Cookies innehåller ofta data som laddats från en server
- Med JavaScript kan man ladda in tidigare cookies samt uppdatera eller spara en ny
- Cookies gör det möjligt att behålla data lokalt även om sidan skulle laddas om
- Viktigt
    - I vissa browsers fungerar cookies endast om koden körs i en webbserver (ex. Chrome)
    - Enkel webbserver kan startas i Python genom att köra:
      $ python3 -m http.server

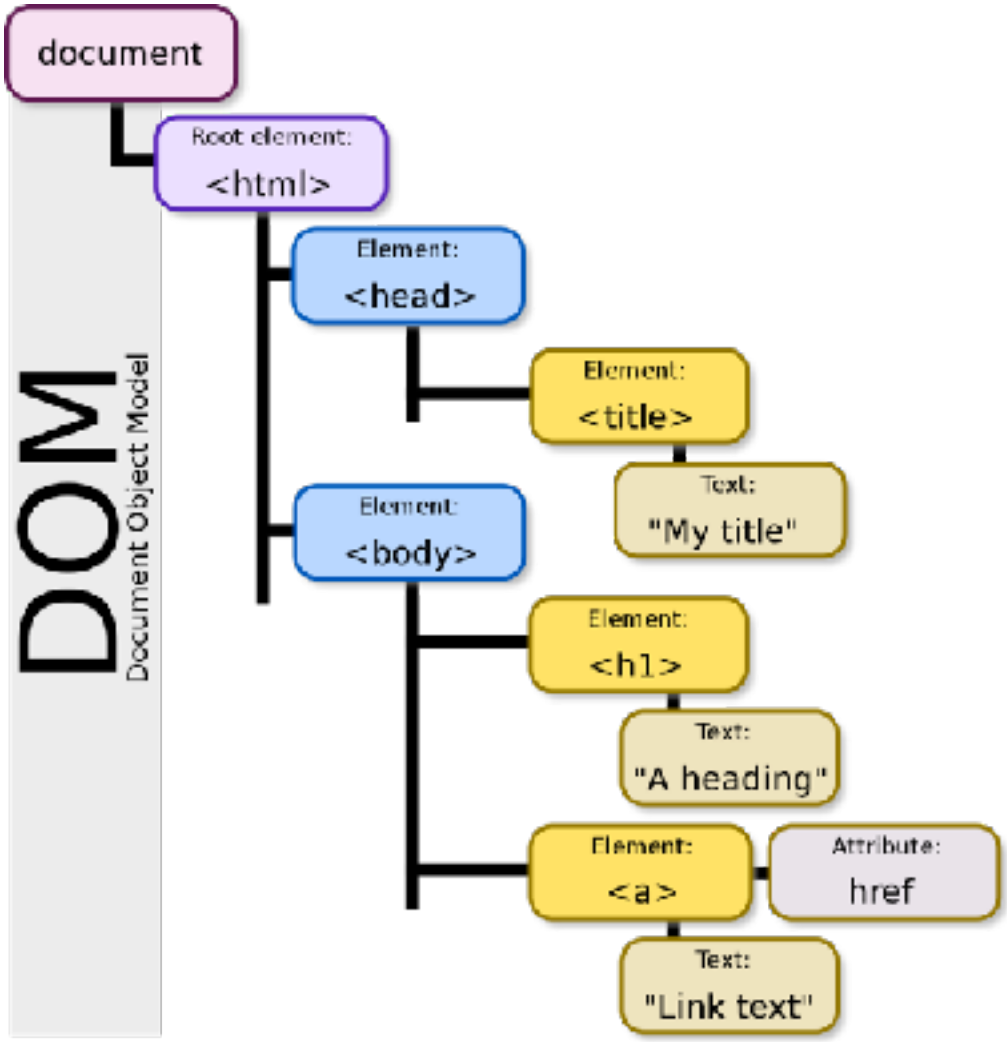
Cookies

- Cookies från klienten skickas med när du gör ett HTTP-anrop till din server
- Används ofta för autentisering
- Cookies **bör** inte innehålla någon känslig information!
- Path-argumentet kan användas för att begränsa när din cookie ska skickas med

Document object model (DOM)

Hemsida





https://en.wikipedia.org/wiki/Document_Object_Model

Noder

- Varje element i ett HTML dokument är en nod i DOM-trädet (inklusive **<!-- kommentarer -->**)
- Det finns 12 olika typer av noder
- Element, TextNode och AttributeNode är de tre typer som generellt sett är intressanta för webbdesign

Navigera i DOM

- document.getElementById('param') returnerar elementet med angivet ID.
- document.getElementByTagName('param') returnerar en lista med element med en viss tag
- document.querySelector(<css selector>) returnerar det fösta elementet baserat på en CSS selector.
- document.querySelectorAll(<css selector>) hämtar en lista av element baserat på en CSS selector.

Operationer på noder

- element.childNodes returnerar en lista med alla noder direkt under element i DOM-trädet.
- element.parentNode returnerar den nod som finns direkt ovanför element i DOM-trädet.
- element.nextSibling returnerar den nod som finns direkt till höger och på samma nivå som element i DOM-trädet.
- element.previousSibling returnerar den nod som finns direkt till vänster och på samma nivå som element i DOM-trädet.

Operationer på noder

- document.createElement('param') skapar ett nytt element baserat på en tag uttryckt som en sträng.
- document.createTextNode('param') skapar en ny TextNode från en sträng.
- element.appendChild(child) placerar det angivna elementet child sist i listan av noder direkt under element.
- element.removeChild(child) tar bort ett element från listan av noder direkt under det specificerade elementet. Noden måste finnas i listan över elementets barn.