JavaScript,

utan tjafs

Anders Strömberg, september 2024

INTRODUKTION

Historik

JavaScript skapades av Brendan Eich 1995. Ett år senare släpptes den första versionen till webbläsaren Netscape 2, och det var då JavaScript 1.0. Netscape lämnade över JavaScript till ECMA (European Computer Manufacturers Association) som idag heter Ecma International. Det är en organisation som skapar och upprätthåller standarder för bland annat programmeringsspråk. De har en stor lista där standarden för skriptspråk finns med. Den heter ECMA-262. Den kallas också för ECMAScript. Skillnaden mellan JavaScript och ECMAScript är alltså att JavaScript är ett skriptspråk som följer specifikationen enligt ECMAScript.

Det kommer nya versioner av ECMAScript ungefär varje år sedan 2015 där det kommer nya uppdateringar i standarden. Äldre webbläsare kan ibland inte hantera dessa nya uppdateringar så därför kan det förekomma diskussioner om vilken version som man bör använda. Använder du ES6 (ECMAScript 6) eller ES2015 som det också kallas eftersom det kom 2015 så kan nästan alla moderna webbläsare hantera det. Den senaste versionen som har kommit är ES15 eller ES2024. För det vi ska använda här är ES6 fullt tillräckligt.

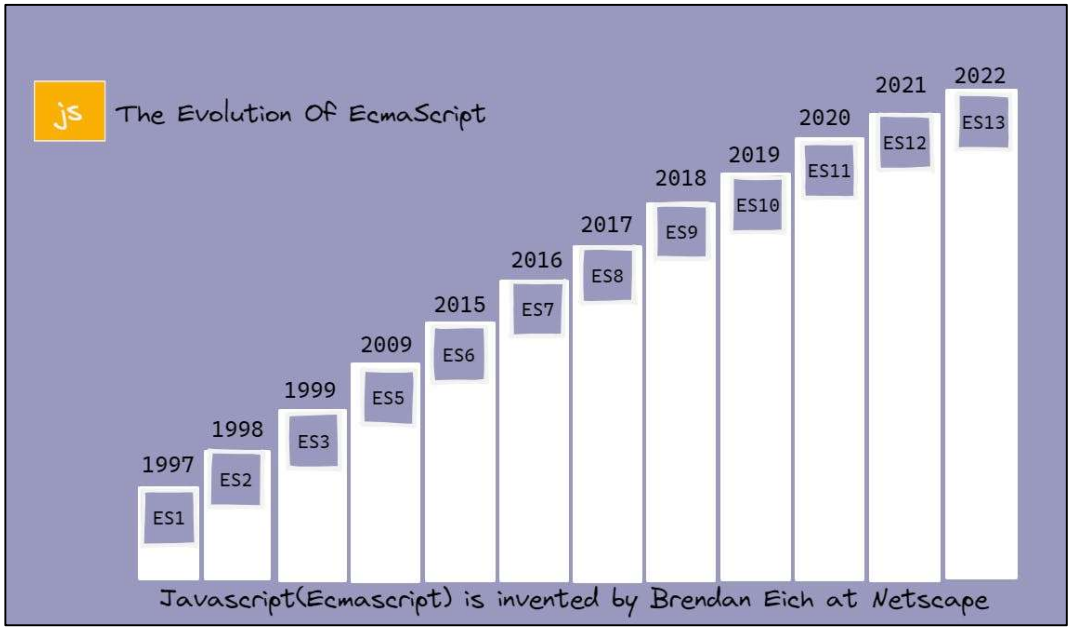


Bild 1 Utvecklingen av ECMAScript genom åren.

Generellt

Det finns som vanligt i programmering flera olika sätt att skriva en och samma lösning på. All JavaScript skrivs alltså i en HTML-tag som heter <script>. Den har en start- och slut-tag och mellan dessa skrivs all kod in. Man kan välja att skriva in koden direkt i HTML-dokumentet. Då kallas det ”inline”, eller lägga det i en separat fil. Vi kommer uteslutande att lägga all vår kod i en separat fil, vilket dessutom är det vanligaste förfarandet. I bild 2 nedan ser vi exempel på JavaScript inline i en fil, vilket är tillåtet men inte rekommendabelt. Det blir mycket mer strukturerat och dessutom lättare att följa om HTML, CSS och JS är separata filer.

```
<body>
  <h1>NAMN, en kanin</h1>
  <button>Ändra färgen</button>
  <p class="namn">Det här är <span>NAMN</span></p>
  <p class="beskrivning">Han är en <span>vit</span> kanin</p>

  <img src="kanin.png" alt="kanin" width="606" height="454" />
  <script>
    const NAME = "Mikael"
    let times = 4

    for (let i = 0; i < times; i++) {
      console.log("My name is " + NAME)
    }
  </script>
</body>
```

Bild 2 JavaScript inline i en .html-fil.

I bild 3 ser vi exempel på en länkning till en separat .js-fil, vilket är det rekommenderade sättet att skriva JavaScript på. I kursen kommer vi ändå att prova på bägge sätten att skriva så att det känns bekant.

```
<body>
  <h1>NAMN, en kanin</h1>
  <button>Ändra färgen</button>
  <p class="namn">Det här är <span>NAMN</span></p>
  <p class="beskrivning">Han är en <span>vit</span> kanin</p>

  <img src="kanin.png" alt="kanin" width="606" height="454" />
  <script src="main.js" defer></script>
</body>
```

Bild 3 JavaScript länkad till en separat .js-fil från en .html-fil.

JavaScript har mycket gemensamt med andra programmeringsspråk såsom Java, C++ och C#. Du kommer därför känna igen mycket från dessa språk. For-loopar, if-satser, while-satser och switch-case-satser är i princip helt identiska.

I JavaScript kan man avsluta varje rad med ett semikolon ; om man vill, men det behövs inte. Den här kursen kommer inte att använda semikolon i de kodexempel som tas upp. Det går att använda både vanliga apostrofer och citationstecken runt strängar. Båda varienterna är tillåtna. Kursen kommer att använda sig av citationstecken. Se bild 4 för exempel på både strängar med citationstecken och apostrofer.

```
let myName = "Thomas"
let yourName = 'Sally'
```

Bild 4 Både citationstecken och apostrofer är tillåtna runt strängar.

Datatyper

De datatyper som förekommer oftast och används i den här kursen är:

- string
- number
- boolean
- object

Förutom dessa kan en variabel också vara null och undefined. Man kan dessutom skapa egna datatyper, men det kommer vi inte använda oss av i den här kursen.

När man deklarerar en variabel i JavaScript finns det två typer som man ska använda sig av, ”let” och ”const”. Den förstnämnda används för en variabel där man vet att den kan ändras, och den andra där man vet att den inte

kommer att ändra sig, under programmets körning. I princip kan du skriva ”let” på alla variabler utan att få ett fel, men det är bra att ta för vana att använda ”const” där man redan under programmeringen vet att variabeln inte kommer att ändra värde. Se bild 5 för deklarationer av några vanliga datatyper.

```
// Numbers:
let length = 16
let weight = 7.5
const TOTAL_WIDTH = 144.0

// String:
let lastName = 'Smith'
const CITY = "Stockholm"

// Booleans
let x = true
let y = false

// Object:
const person = { firstName: "John", lastName: "Doe" }
```

Bild 5 Deklaration av vanliga datatyper i JavaScript.

Själva deklarationen av variabeln innebär att man använder sig av let eller const och därefter namnet på variabeln. Man sätter inte ut vilken typ av variabel det ska vara. Det känner JavaScript av själv. Enligt bild 3 har de tre första variablerna datatypen number. Det kan vara både heltal och decimaltal, allt går in under number. Den variabeln som är deklarerad med const behöver inte ha ett namn i versaler, men det är vanligt förekommande att man använder sig av det ändå. I den här kursen använder vi den varianten. Det finns dessutom en tredje typ av deklaration som fortfarande är tillåten, men inte rekommenderad att använda sig av längre. Det är ”var”, som användes tidigare, och som du möjligen kan stöta på när du läser gammal kod. I den här kursen använder vi bara ”let” och ”const”.

Listor och objekt

En array eller lista ser ut som på bild 6. Lägg märke till hakparenteserna som omger listan.

```
// Lista med strängar:
const cars = ["Saab", "Volvo", "BMW"]
```

Bild 6 En lista med strängar.

En lista kan vara en lista av strängar som på bild 6, eller en lista av booleans, numbers, objects eller egendefinierade datatyper.

På bild 7 ser man ett objekt. Ett objekt är omgivet av klammerparenteser eller ”måsvingar”. Objektet kan ha flera attribut som kan vara av vilken datatyp som helst. I exemplet nedan är det strängar, tal och booleska värden. Det går att använda attribut som är egna listor, underobjekt eller egna datatyper.

```
// Objekt med fyra attribut:
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 28,
  married: false
}
```

Bild 7 Ett objekt med fyra attribut, firstName, lastName, age och married.

Funktioner

Funktioner skrivs i den här kursen enligt bild 8.

```
function greeting() {
  console.log("Hello")
}
```

Bild 8 Exempel på hur man skriver en funktion i JavaScript.

Genom att använda nyckelordet function berättar man för JavaScript att här vill jag skriva en funktion. Efter det kommer namnet på funktionen. Allt man vill att funktionen ska göra skrivs i funktionskroppen som består av måsvingarna.

I JavaScript så kan man också skriva funktioner enligt en ”arrow function” eller som en ”anonymous function”. På bild 9 visas en och samma funktion på tre olika sätt. Kom ihåg att i den här kursen skriver vi enligt variant 1.

```
// Variant 1 - plain function
function greeting() {
  console.log("Hello")
}

// Variant 2 - arrow function
const greeting = () => {
  console.log("Hello")
}

// Variant 3 - anonymous function
const lightbulb = document.getElementById("lightbulb")
lightbulb.addEventListener("click", function () {
  console.log("Hello")
})
```

Bild 9 Tre exempel på hur man skriver en funktion i JavaScript.

DOM-trädet

När en webbsida laddas, skapar webbläsaren en DOM (= Document Object Model). Det är en data-representation av webbsidan. Den är i princip uppbyggd som ett upp- och nervänt träd med roten längst upp. Se bild 10 för ett exempel på ett DOM-träd.

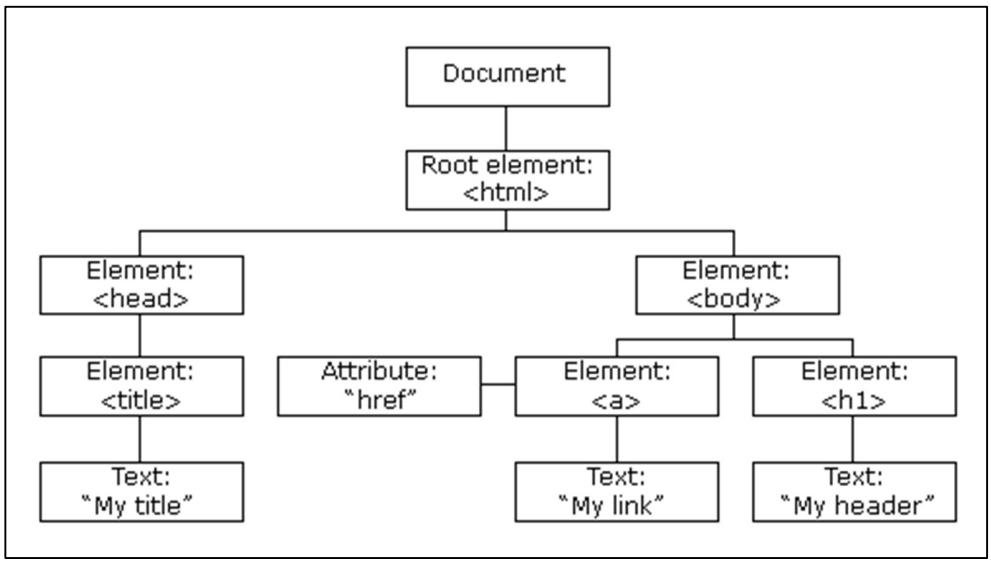


Bild 10 Ett exempel ett DOM-träd av en webbsida.

Det går alltså att manipulera och ändra egenskaperna på alla taggar på en webbsida genom JavaScript. Nästlade taggar är barn-taggar till den omslutande taggen. Texten inuti en tagg är också ett objekt. Till exempel så är ”document.body” det objekt som representerar taggen <body>. Om man vill ändra bakgrunden på den taggen skriver man enligt bild 11.

```
document.body.style.background = 'green'
```

Bild 11 Exempel på hur man kan ändra bakgrundsfärgen till grön på body-taggen.

Det bästa och enklaste sättet att komma åt en tagg eller ett element är att sätta det till en variabel och därefter manipulera den så som man vill. För att kunna hämta den går det att använda flera kommandon. De vanligaste och smidigaste är de följande och det är också dessa som förekommer i kursen. Men det finns fler varianter som man kan utforska när man är lite varmare i kläderna. På bild 12 ser vi ett stycke HTML-kod med en tagg som vi vill komma åt och göra lite saker med. Det bästa sättet är att att förse taggen med ett unikt id för att sedan i JavaScript kunna hämta just det elementet.

```
<div id="box">En låda</div>
```

Bild 12 En div-tagg med ett unikt id.

I JavaScript kan man nu hämta div-taggen från bild 11 genom att använda kommandot ”document.getElementById()”. Se bild 13 för exakt syntax av kommandot.

```
const boxTagg = document.getElementById("box")
```

Bild 13 Syntax för hur man sparar en tagg till en variabel.

Vi sparar taggen som är en till en konstant variabel som kallas ”boxTagg”. Efter det här kan vi lägga till attribut eller ändra texten i taggen. Enligt bild 14 så ändrar vi textfärgen till röd och lägger till en svart ram runt taggen.

```
const boxTagg = document.getElementById("box")
boxTagg.style.color = "red"
boxTagg.style.border = "1px solid black"
```

Bild 14 Exempel på hur man ändrar textfärg och lägger till en ram med JavaScript.

Vad är TypeScript

I den här kursen kommer vi inte att jobba alls med TypeScript, men det är ändå bra att känna till dess existens. TypeScript är ett starkt typat programmeringsspråk som bygger på JavaScript. Den stora nackdelen med JavaScript är att det är ett språk som är löst typat. Det innebär att en variabel kan ändra skepnad under programkörningen. Se bild 15 för ett exempel på en variabel som ändrar sig.

```
let times = 4
times = "hejsan"
times = false
```

Bild 15 En läskig variabel som ändrar skepnad.

När variabeln times deklareras på första raden är den av typen number. På den andra raden ändras den till en sträng för att slutligen bli en boolean. Det här är fullt tillåtet i JavaScript. Bara för att det är tillåtet så innebär det inte att det är bra, vilket det inte är.

I ett mindre program så har man ganska bra koll på sina variabler och vad det är tänkt att de ska vara för typ, men i ett lite större program med flera .js-filer blir det snabbt svårt att överblicka. För att överbrygga den här problematiken så uppfanns språket TypeScript. Där kan man tvinga variabler att vara en och samma typ genom hela programmet. I utvecklingsfasen av programmet så hjälper TypeScript till att varna för felaktiga datatyper på variabler, parametrar och funktioner. När man har skrivit ihop sina .ts-filer så kompilerar TypeScript-servern om dessa filer till .js-filer vilka ser ut som helt vanliga filer.

På bild 16 ser vi hur variabeln firstName bestäms till datatypen string. På raden under försöker programmeraren sätta datatypen till number genom att sätta den till 33. I det här skedet kommer TS-servern att visa att det inte går genom att markera firstName med ett rött understreck.

```
let firstName: string = "Dylan"
firstName = 33
```

Bild 16 En variabel som är förutbestämd att vara en sträng kan inte ändras i TypeScript.

Koppling mellan HTML och JavaScript

JavaScript lägger man till i en tag som heter <script>. Den behöver en start- och en sluttagg. Man kan antingen lägga till den inline, det vill säga i själva HTML-filen. Om man bara har en eller några få rader kan det vara okej. Det vanligaste är att man kör JavaScript i en separat fil och då lägger man till en script-tagg där sökvägen till JavaScript-filen står. Man använder en script-tagg för varje JavaScript-fil man vill lägga till. Se bild 17 för skillnaden mellan så kallad inline script, och en separat JS-fil.

```
// Inline script
<script>
    "use strict"

    console.log("hello world")
</script>

// Länk till en separat fil
<script src="main.js" defer></script>  // HTML-kod

"use strict"                           // JavaScript-kod
console.log("hello world")
```

Bild 17 De två olika sätten att inkludera JavaScript till en hemsida.

När en webbsida laddas så läser den in alla taggar uppifrån och ner. Om det är några taggar som man behöver komma åt och använda i sin JavaScript-kod så vill man vara säker på att de först är inlästa och definierade i sin HTML-kod. Det är lätt att få något felmeddelande om man försöker komma åt taggar som inte har hunnit bli inlästa innan man försöker komma åt dem i sin JavaScript. För att komma undan det här så är det dels en bra idé att lägga till script-taggen i sin HTML så långt ner som möjligt innanför body-taggen. Man kan också lägga till nyckelordet defer, vilket innebär att man säger till JavaScript att vänta med att köra tills hela html-sidan är laddad. Se bild 18 för ett exempel på hur man bör placera sin script-tagg.

```
      </footer>

    <script src="main.js" defer></script>
  </body>
```

Bild 18 Placering av slut-script-taggen i ett html-dokument.

”use strict”

Det finns ett tillägg som man kan använda sig av i toppen av alla JavaScript-koder som faktiskt rekommenderas. Det är att skriva ”use strict” överst, alltså på första raden av alla js-filer. Då säger man till JavaScript-motorn att vara extra ”petig” med kontrollen av det som skrivs och man får lite extra hjälp. I kursen kommer vi att använda oss av det, men det är inte säkert att man ser det i alla koder man läser. Enligt bild 19 ser vi att ”use strict” används. Längre ner i skriptet sätts variabeln firstName till värdet ”Kalle”. Problemet här är att firstName inte är deklarerad med let eller var. Om man inte använder ”use strict”-direktivet kommer man inte att få någon varning från JavaScript-motorn, men det får man när det används.

```
"use strict"

firstName = "Kalle"
```

Bild 19 Användning av ”use strict” överst i js-filen.

console.log

Det är svårt med JavaScript i början, speciellt att ha koll på vad som händer i koden och i vilka funktioner man befinner sig. Därför kommer kursen att använd console.log som är en funktion för att skriva ut värden i konsolfönstret. I kursen kommer vi att använda oss av utskrifter till konsolen för att kontroller vilka värden variabler har i vissa lägen. För att få fram konsolen går man till webbläsaren och trycker på tangenten F12 när man har fliken aktiverad. På bild 20 nedan kan man se i webbläsarens högra del det som kallas ”developer tools”. Den öppnas oftast till höger i webbläsaren. Högst upp i developer tools finns det några flikar, bland annat en som heter Console. Klicka på den så får man upp konsolfönstret. Här kan man se utskrifter som görs i JavaScript-koden.

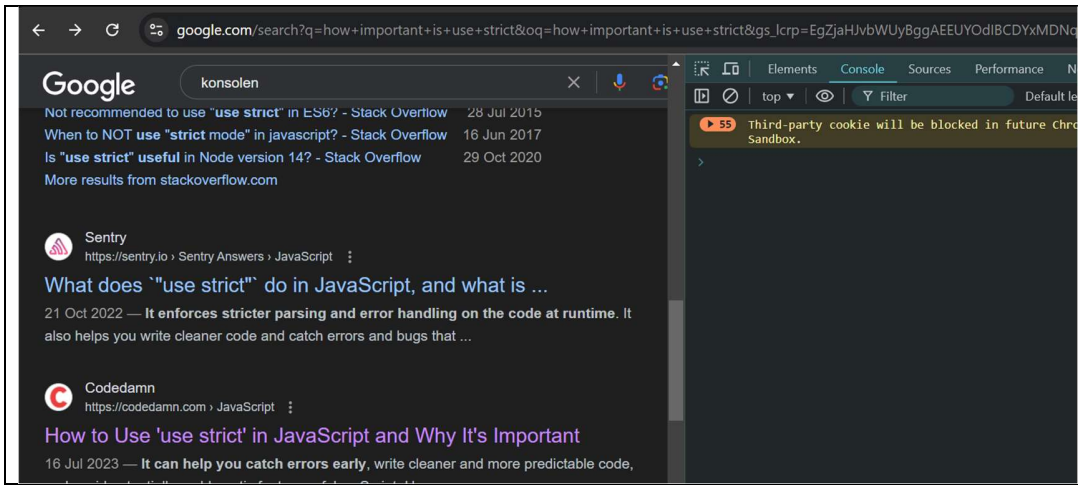


Bild 20 Konsolfönstret i developer tools.

Bild 21 visar hur ett console.log -kommando ser ut i kod respektive hur det ser ut konsolfönstret.

```
console.log("hello world")
```

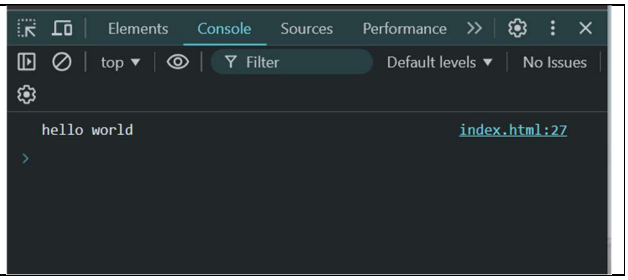


Bild 21 Koden för att skriva ut en sträng respektive konsolfönstret med själva utskriften.

Kodstruktur

Det kan lätt bli lite stökig kod när man skriver JavaScript så därför är det viktigt att ha en bra ordning och struktur på koden. I stora drag så är det bra att ha följande struktur på koden (se bild 22).

```
"use strict"

// Variabler och konstanter som används i programmet
// ================================================================
const knapp = document.querySelector("button")
const rubrik = document.querySelector("h1")
const djur = document.getElementById("djur")
// ================================================================


// Funktioner som används genom programmet
// ================================================================
function tjena() {
  console.log(”tjena”)
}

function hej() {
  console.log(”hej”)
}
// ================================================================


// Eventlisteners och funktionsanrop
// ================================================================
knapp.addEventListener("click", tjena)
// ================================================================
```

Bild 22 Ordning och struktur i js-filen.

Kursupplägg

I kursen kommer du att få jobba med fem olika projekt på egen hand. Där vi tillsammans går igenom syftet med projektet, alltså vad som ska göras, hur vi ska göra det och tittar på koden för att förstå de olika stegen.

- 01-koppling
  I det första projektet blir det en mjukstart. Vi ska titta på kopplingen mellan dokumenten. Hur skriver man i HTML för att få JavaScript att fungera. Både inline och som en extern .js-fil.

- 02-kaninfärg
  Här kommer vi att byta färg på en kanin genom olika knappar. Vi tittar på ett ”click event” som sätter olika fäger beroende på vilken knapp man trycker på.

- 03-glödlampa
  Vi ska tända och släcka en glödlampa, men vi har bara en knapp. Hur ska vi bära oss åt för att få det att fungera?

- 04-slumpdjur
  Här ska vi slumpa fram ett älsklingsdjur och ett favoritnamn på det djuret. Vi kommer återigen att jobba med ”click event” och listor.

- 05-sötisgalleri
  Vi avrundar med ett galleri med söta djur. Vi vill kunna se djurbilderna i lite större format när vi klickar på de små bilderna.

0. KOMMA IGÅNG – FIX OCH DON

Hämta kursmaterial

Innan vi kan börja med någon kod över huvud behöver vi skapa en arbetsyta för de projekt vi ska jobba med i senare delar.

1. Skapa en ny mapp där du vill jobba med den här kursen, förslagsvis js-kurs

2. För att kunna hänga med så behöver du gå till ett GitHub-repo som finns på följande länk https://github.com/anst9000/js-kursmaterial

3. När du kommer till den sidan så går du till en grön knapp med namnet '<> Code' klickar på den och väljer alternativet 'Download ZIP'.

4. Packa upp (unzip) filerna i den katalog du vill använda för den här kursen (js-kurs).

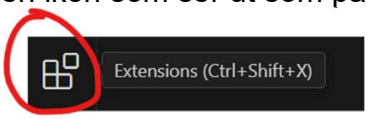
5. Öppna nu VS Code och gå till rullgardinsmenyerna högst upp.

6. Välj File -> Open Folder , och välj den mapp som du har skapat för den här kursen.

7. Det borde finnas 5 mappar:

- 01-koppling
- 02-kaninfärg
- 03-glödlampa
- 04-slumpdjur
- 05-sötisgalleri

8. För att kunna jobba med hemsidan är det lämpligt att köra den lokalt på en egen webbserver. I VS Code längs ute till vänster finns en ikon som ser ut som på bilden.



9. Klicka på ikonen och skriv i sökrutan högst upp i Extensions, **Live Server** och välj sedan installera.

10. När det är klart kan du öppna filen ’index.html’ och högerklicka.

11. Välj alternativ ’Open with Live Server’

12. Nu borde hemsidan visas i en ny flik i webbläsaren på adressen http://127.0.0.1:5500/index.html

13. Gör den inte det så kopierar du adressen och öppnar en ny flik och klistrar in den där, eller skriver ’localhost:5500’ för hand. Motsvarigheten till 127.0.0.1 är localhost. Så skrivsätten http://127.0.0.1:5500 , 127.0.0.1:5500 och localhost:5500 går alla lika bra (man måste inte skriva ut http://)

14. För varje nytt projekt vi börjar jobba med så stänger vi först alla öppna flikar i editorn. Därefter går vi till rätt projektmapp och öppnar den. Vi vill öppna index.html först och högerklicka för att kunna starta live server.

1. PROJEKT 01-KOPPLING

Kopplingen mellan JavaScript och HTML

I det här projektet ska vi se till att kopplingen mellan HTML-koden och JavaScript fungerar. Men vi ska också göra sidan lite trevligare genom att lägga till kopplingen till CSS-filen. Vi börjar med att titta på de filer som ligger i mappen. Det är sammanlagt fyra filer, men två av dem behöver du inte jobba med. Du kan öppna style.css och spelling_love.jpg och få en uppfattning om dem. Filen main.js är helt tom. Den ska vi fylla med lite innehåll. På bild 23 ser du hur filen index.html ser ut innan du har lagt till något innehåll i den.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width,
     initial-scale=1.0" />
    <title>01-koppling</title>
  </head>
  <body>
    <h1>01-koppling</h1>

    <img
      src="spelling_love.jpg"
      alt="winnie and piglet"
      width="1400"
      height="1967"
    />
  </body>
</html>
```

Bild 23 Innehållet i filen index.html.

Du kan starta live server genom att högerklicka i html-filen och se till att du kan se den i en webbläsare. Sidan är gigantisk eftersom det är en jättestor bild som visas. Det kommer att se mycket bättre ut när vi kopplar css till html-filen. Så det blir första steget. Inne i head-taggen kan du lägga till länken till style.css. Om du inte kommer ihåg hur man skriver kan du tjuvkika på bild 24.

```
<head>
    <link rel="stylesheet" href="style.css" />
</head>
```

Bild 24 Länken till style.css som bör ligga inne i head-taggen.

Om du har lyckats bör sidan se mycket bättre ut än innan. Nu ska vi försöka lägga till JavaScript. Men för att kontrollera att det blir som vi har tänkt får du gå till webbläsaren och aktivera fliken och därefter klicka på F12 (det är developer tools). Det går också att få fram developer tools genom att högerklicka och välja inspektera som brukar ligga längst ner av alla alternativ. I developer tools klickar du på fliken Console. Nu är det dags att lägga till JavaScript på sidan.

Börja med att gå till filen index.html och gå längst ner i body-taggen, men innan du kommer till sluttaggen </body>. Nu skriver du en start- och sluttagg script. Därefter lägger du till direktivet ”use strict” och gör en konsolutskrift av ett valfritt textmeddelande. På bild 25 ser du hur hela html-filen bör se ut efter att både css och js är tillagt.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width,
     initial-scale=1.0" />
    <link rel="stylesheet" href="style.css" />
    <title>01-koppling</title>
  </head>
  <body>
    <h1>01-koppling</h1>

    <img src="spelling_love.jpg"alt="winnie and piglet"
      width="1400" height="1967"
    />

    <script>
      "use strict"

      console.log("hello world")
    </script>
  </body>
</html>
```

Bild 25 Innehållet i filen index.html.

För att kontrollera att JavaScript har blivit korrekt tillagt så kan vi gå till webbläsaren och kolla i konsolfönstret. Om du ser din text utskriven i fönstret

vet du att allt har blivit rätt och riktigt. Kan du inte se utskriften kan du prova ladda om sidan. Det bästa sättet att ladda om sidan är genom att att använda CTRL+F5, för då rensar man eventuella data som har blivit sparat lokalt på datorn (cache) och man ber webbläsaren att hämta alla filer från din lokala server. Om du fortfarande inte får fram din utskrift får du gå tillbaka till filen index.html och kontrollera att du inte har skrivit fel vid script-taggen.

Om det gick bra att skriva JavaScript så att du fick fram ett konsolmeddelande i developer tools så har du gjort det mycket bra! Nu ska vi prova att skriva en helt separat fil och istället länka till den. Byt ut innehållet som du hade i script-taggen i filen index.html enligt bild 26.

```
<!-- Före -->

<script>
  "use strict"

  console.log("hello world")
</script>
```

```
    <!-- Efter -->

<script src="main.js">
</script>
```

Bild 26 Script-taggen i filen index.html före och efter ändringen.

Nu hämtar vi helt enkelt innehållet i script-taggen och klistrar in det i filen som heter main.js istället. Om vi nu går tillbaka till konsolfönstret så bör vi se samma utskrift där som innan.

2. PROJEKT 02-KANINFÄRG

Interagera mellan HTML och JavaScript

Nu är det dags att göra lite mer med JavaScript. Vi ska se vad som händer när vi klickar på en knapp, hur vi kan få saker att hända när vi har en så kallad eventlistener. När en specifik händelse inträffar ska vi utföra några saker. Vi börjar med att gå till projekt 02-kaninfärg och öppna filen index.html. Den bör se ut som på bild 26 nedan.

Du kan öppna style.css och spelling_love.jpg och få en uppfattning om dem. Filen main.js är helt tom. Den ska vi fylla med lite innehåll. På bild 27 ser du hur filen index.html ser ut innan du har lagt till något innehåll i den.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width,
     initial-scale=1.0" />
    <link rel="stylesheet" href="style.css" />
    <title>02-kaninfärg</title>
  </head>
  <body>
    <h1>NAMN, en kanin</h1>
    <button class="farg-knapp">Ändra färgen</button>
    <p class="beskrivning">Det är en
       <span>vit</span> kanin
    </p>

    <div id="kanin">
      <img src="kanin.png"
           alt="kanin"
           width="606"
           height="454"
       />
    </div>

    <script src="main.js"></script>
  </body>
</html>
```

Bild 27 Innehållet i filen index.html.

I den här filen är både style.css och main.js redan länkade till. Så nu kan vi starta vår lokala webbserver genom att högerklicka i filen index.html och

starta live server. Om allt är rätt och riktigt ska du nu få syn på en söt kanin. Här är det mycket viktigt att vi fyller i rätt namn på kaninen. I filen index.html Så står det som rubrik NAMN, en kanin. Du måste fylla i kaninens namn (välj ett passande namn själv) och skriva in det i samma fil på taggen h1. Nu kan vi återgå till mindre viktiga saker. Det är några saker som också är bra att lägga märke till i den här filen. För att kunna ”jobba” med olika element från DOM-trädet i JavaScript är det ofta en fördel att namnge sina element eller taggar med ett id=”kanin” eller id=”andraFargKnapp” så att det inte blir knepigt att hitta rätt element senare. Det kan ju finnas fler <button> och flera <h1>. Det är ju också tillåtet att använda class=”btn” på flera olika element, men vi vill ju vara säkra på att vi hittar rätt element. För id får det bara finnas ett element med det id-namnet. Vi kommer ändå att träna på att hämta olika typer av element så här går det ändå bra.

Vi kommer inte behöva använda eller någonting i filen style.css, men det är ändå okej att öppna och titta i den på egen hand. Däremot kommer vi att jobba med filen main.js eftersom det är JavaScript vi lär oss. Den filen ser i grundutförande ut som på bild 28.

```
"use strict"

function andraFarg() {
  const allaFargerLangd = 5
  const slumptal = Math.floor(Math.random() * allaFargerLangd)
  const allaFargerEn = [
    "yellow",
    "white",
    "pink",
    "lightblue",
    "darkgrey",
  ]
  const allaFargerSv = ["gul", "vit", "rosa", "ljusblå", "grå"]
  const nyFargEn = allaFargerEn[slumptal]
  const nyFargSv = allaFargerSv[slumptal]
}
```

Bild 28 Innehållet från start i filen main.js.

Det som för tillfället finns i den här filen är just nu två saker. Dels längst upp på sidan ”use strict” och en funktion som heter andraFarg. Tanken är att funktionen ska ändra färg på kaninen när man trycker på den gula knappen längst upp på webbsidan. Vi ska också ändra vilken färg texten säger att kaninen har. Därför kommer vi att behöva tre så kallade handtag till dessa html-element. Ett handtag för knappen, ett för texten där det just nu står ’vit’ och ett för den div-tagg som innesluter vår kaninbild. Så det första steget blir därför att skapa dessa handtag.

Gör en liten kommentar någon rad under ”use strict” som vi kallar // Variabler.

Nu gör vi först en variabel till knappen som har en klass ”farg-knapp”, en för texten vit som ligger i en span innanför p med klassen ”beskrivning” och en för taggen som innesluter kaninbilden och den har ett id ”kanin”. Det kan se ut som i bild 29.

```
"use strict"

// Variabler
const fargKnapp = document.querySelector(”.farg-knapp”)
const fargTextSv = document.querySelector(”.beskrivning span”)
const kaninBakgrund = document.getElementById(”kanin”)

// Funktioner
function andraFarg() {
}

// Event listeners
```

Bild 28 Innehållet från start i filen main.js.

Vi kommer behöva komplettera funktionen andraFarg lite grann, men det tar vi sist av allt. Nu är det dags att registrera en event listener till knappen. Den ska specifikt lyssna på click events, alltså knapptryckningar. När man trycker på knappen ska den kalla på funktionen andraFarg(). Det kan man skriva som i bild 29.

```
// Event listeners
fargKnapp.addEventListener(”click”, andraFarg)
```

Bild 29 En event listener för ett click event.

För att kontroller att det verkligen fungerar så lägger vi till en console.log(”hello”) högst upp i vår funktion andraFarg enligt bild 30. Vi kan kommentera bort allt annat, eller låta det stå kvar, det påverkar inte utfallet.

```
function andraFarg() {
  console.log(”hello”)
}
```

Bild 30 En console log för att kontrollera att vårt click event fungerar.

Starta nu upp live server genom att gå till filen index.html och högerklicka. Gå därefter till webbläsaren och öppna upp developer tools genom att trycka på F12. Markera fliken ”Console”. Om det fungerar ska det nu dyka upp hello när man klickar på knappen.

För att allt ska fungera måste vi lägga till en rad där vi byter text ”vit kanin” till den färgen kaninen verkligen är och ändrar bakgrundsfärgen till den färgen

den verkligen ska ha. Om man kikar på funktionen andraFarg lite närmare så ser man att den har två listor, allaFargerEn och allaFargerSv. Det är för att hålla reda på färgerna på engelska och svenska. Sedan slumpas det fram ett nummer som kan användas för att hämta en slumpad färg på engelska och svenska. De kompletterande raderna i funktionen blir därför enligt bild 31.

```
function andraFarg() {
  const allaFargerLangd = 5
  const slumptal = Math.floor(Math.random() * allaFargerLangd)
  const allaFargerEn = [
    "yellow",
    "white",
    "pink",
    "lightblue",
    "darkgrey",
  ]
  const allaFargerSv = ["gul", "vit", "rosa", "ljusblå", "grå"]
  const nyFargEn = allaFargerEn[slumptal]
  const nyFargSv = allaFargerSv[slumptal]

  kaninBakgrund.style.backgroundColor = nyFargEn
  fargTextSv.innerHTML = nyFargSv
}
```

Bild 28 Innehållet från start i filen main.js.

Vi ändrar style.backgroundColor på elementet kaninBakgrund och sätter den lika med den nya engelska färgen. Dessutom ändrar vi innerHTML på elementet fargTextSv och sätter den lika med den nya svenska texten.

3. PROJEKT 03 -> PROJEKT 05

Görs på egen hand

Projekt 03, 04 och 05 går att öppna upp och göra på egen hand. Prova att kommentera bort eller ta bort delar från koden och skriv det du klarar av på egen hand.