Anteckningar Javascript

# Dag 1 2020-11-16

# Programeringsparadigm

Programmeringsspråk kan fungera på olika vis.

Java script tenderar att vara objektorienterad eller funktionell. Med detta menas att koden är strukturerad på ett vist vis och olika ansvar kan delas upp till olika delar utav koden.

En skillnad mellan JavaScript och C# är att JavaScript kompileras samtidigt som koden körs. Dvs om du har ett syntax fel så märks inte det förens att du försöker köra koden.

### Svagt typat

JavaScript är ett skriptspråk. Det är även svagt typat vilket menar att koden och värden i den kan ändras då den körs.

**Var x = 10;**

**Console.log(x);**

**X = ”Hej”** Här försöker jag byta data typ då programmet körs. Det hade ej funkat i C# men det är ok i JavaScript.

## Variabler

**Var siffra = 10;**

Precis som C#.

* Var = data typ (var är en själv tilldelade data typ. Dvs den skapar den data typen som är lämplig beroende på dess värde).
* Siffra = identifierare, namnet vi ger vår variabel.
* = tilldelningsoperator
* 10 = värde
* ; man måste inte alltid avsluta med ett semikolon men det är bra standard att göra.

### Let

Let är en variabel som finns i C# och den är rekommenderad över var.

Skillnaden mellan **let** och **var** är att var kan påverka saker utanför scope trots att den inte ska det. **let** gör inte det så använd **let**.

**Char** finns inte i JavaScript.

**Decimal** finns inte i JavaScript.

Du anger aldrig data typen i JavaScript utan du använder **let** som gör det åt dig.

Om du ger en variabel värdet null så blir data typen **object.**

**Undefined** är en data typ som uppstår när du antagligen tilldelar en variabel **undefined** eller inte tilldelar en variabel något värde.

## Lista i JavaScript

En skillnad mot c# är att i listor skapade i JavaScript är att du kan lägga in olika typer utav variabler. Du kan också placera objekt i listor. Du behöver inte ange hur många positioner en lista ska ha heller utan det är bara att fylla på.

**Let list = [**

**”Niklas”,**

**31,**

**True,** Olika Datatyper i samma lista.

**False,**

**{**

**Namn: ”Gustav”,**

**Ålder: 200**

**}**

**];**

## Skapa Objekt

**Let obj = {** jag skapar ett objekt.

**firstName: ”Gustav”,** med 2 variabler.

**age: 31**

**};**

**Let obj2 = obj;** jag skapar ett objekt till och tilldelar det värdet utav det första objektet.

**Obj2.age = 25;** jag ändrar värdet på åldern i objekt 2.

När man tilldelar ett objekt ett värde så tilldelar man egentligen bara des plats i minnet. Så de får samma värde.

Så om man sedan ändrar på värde två så ändrar sig även värde 1 även om man skriver att det är värde 2 man vill ändra. Så att duplicera objekt som jag gjorde ovan är egentligen att skapa två positioner utav samma objekt. När man interagerar med det ena så kommer det andra också att ändra på sig.

## Function/Funktion

En funktion är en metod i JavaScript. Den har alltså samma syfte som en metod i C#. En stycke kod som man vill kunna anropa flera gånger.

**Function loopThroughArray (arr) {**

**For (let = i; i <arr.length; i++)**

**{**

**Const element = arr[i];**

**Console.log(element);**

**}**

**}**

## Operatorer

Operatorer i JavaScript fungerar likadant som i C#.

Så **=** är en tilldelningsoperator. Den anger ett värde.

**==** är lika med.

I JavaScript så kan detta innebära att en string med värdet ”10” är samma som ett numeriskt 10-värde. Med två ekvivalens tecken så respekteras alltså inte datatypen.

**Let string =”10”;**

**Let num = 10;**

**If (string == num)** Denna if-sats skulle returnera true. Alltså köras.

**===** kollar också data typen.

**if (string === num)** Denna if-sats skulle returnera false, och inte köras.

Så en sträng med **”10”** är inte lika med ett numeriskt 10 om du använder tre **=**.

## Villkor

Villkor fungerar nästan likadant som i C#.

En skillnad man kan använda sig utav är:

**If(!obj)** Om ett objekt inte existerar eller har värdet null (samma sak) så blir villkoret true och körs.

**If(0)** Detta skulle returnera false och inte köras.

## Att länka en JavaScript fil i HTML

För att länka en JavaScript fil i en HTML så skriver du följande.

**<script src=”script.js” defer></script>**

Defer ser till att allt läses in i korrekt ordning. Vad den gör är att den låter sidan läsa färdigt innan den applicerar javascriptet.

# Dag 2 2020-11-19

## Objekt

Objekt i JavaScript är ganska så likt en array. Man skapar dem så här:

**Let obj = {**

**Namn:”Dennis”** Nyckel (identifierare/namn, man kallar det nyckel i objekt), kolon, följt utav värde

**Nyckel: false**

**}**

För att skriva ut värdet utav en property (ett värde i ett objekt, jag skulle bara kalla det en variabel) så skriver man **objektnamn.nyckel**.

**Console.log(obj.namn);**

**”Dennis”**

**Console.log(obj.[”namn”]);**

**”Dennis”**

I objekt använder du alltid nyckel/identifieraren. Det går ej att använda index.

## Array/vektor

Let arr = [1, ,20, ”hej”];

Samma som list.

## I JS är till och med funktioner objekt

Funktioner kan ha egenskaper

Funktioner kan ha metoder

Funktioner Kan anropas (till skillnad från andra objekt).

Om man försöker loga en funktion utan parenteserna så skrivs hela funktionen ut. Det är för att det inte tolkas som att du försöker anropa den utan att du vill skriva ut ett helt objekt.

## Callbacks

Är vad en funktion som passas till en annan där den sedan anropas.

**Function ageCheck (value, callback){** här har vi en fuktion som ber om ett värde och en annan funktion som input. **Value** ska vara ett numeriskt värde.

**If (value >= 18){**

**Callback();** här anropas value om vilkoret returnerar true

**}**

**}**

**Function ageOK(){** detta är functionen som vi skickar med.

**Console.log(”You may enter”)**

**}**

**Const ageInput = Document.QuerySelector(input);** vi tar ett värde ifrån en input i HTML (ett textfält användaren fyller i).

**Let age = ageInput.value** vi skapar en variabel som lagrar värdet utav det input.

**ageCheck(age, ageOK)** vi skickar input + vår callback funktion.

## String interolation

I JavaScript kan man använda ett dollartecken $ på varje variabel man vill sätta in om man använder backticks: **`** (shift + ´ + space)

**Let Age = 24;**

**Alert(`you are ${age} years old`)**

**”you are 24 years old”**

Man kan också göra som i C#

**Let textOchÅlder = ”hej du är ” + age + ” år gammal.”**

**Console.log(textOchÅlder):**

**”hej du är 24 år gammal.”**

## Higher order functions

Inbyggda funktioner som tar ett vist värde (olika per funktion) och sedan gör något med det värde och sedan använder en callback funktion på resultatet utav det ursprungliga värdet.

## Filter

En inbyggd funktion som kan användas för att filtrera bort element i ett array enligt ett villkor som du sätter. Om du text har en stor mängd objekt i en array och du vill ha fram dem med ett vist värde. Då kan du använda ett filter.

Den returnerar en ny array med de värden som du ville ha.

**variabel.filter(function(returvariabel){vilkor}))**

**Ex**

**Console.log(name.filter(function(name){return namn === ”David”}))** Denna filter funktion skulle returnera och sedan skriva ut alla med namnet David.

## For each

Om man vill göra något med varje element i en array så kan man använda en foreach funktion.

Ex:

Vi har en array där vi vill skriva ut varje element. Det kan behöva göras flera gånger så vi skapar en callback funktion.

**Let input –[1, 23, 222, 554];** vår array

**Function write(val){** funktionen som skriver ut.

**Console.log(val)**

**}**

**Foreach(input, write)** vi anropar vår funktion. Den går igenom varje element och applicerar funktionen ”write” på varje element.

**Function for each (arr, callback){**

**For (let index = 0; index<arr.lenght; index++){**

**Const element = arr[index];** funktionen går igenom varje element

**Callback(element);** och gör en callback till utskrifts metoden.

**}**

## Reduce

En reduce funktion reducerar en array till ett enda värde. Säg att du vill ta ut summan utav alla värden i en array.

**Const nummbers = [1, 2, 4, 6, 8]**

**Console(nummbers.reduce(function(prevValue, currValue){ return preValue + curValue},0))**

Vi vill aplicera functionen reduce på arrayen ”nummbers” (**nummbers.reduce**).

Vi har två invärden i funktionen (**prevValue, currValue**).

Sedan kommer vad funktionen ska skicka tillbaka(**{return preValue + curValue}**).

Nollan är det intiela värdet på returvariabeln. Alltså vad som finns innan vilkoret appliceras.

## Map

En ”Map funktion”

Att notera:

Jag är inte riktigt med på hur det vi gått igenom idag funkar därav så kan jag inte riktigt anteckna något vettigt. Måste kolla igenom det själv.

# Dag 3 2020-11-23

## DOM, Document Object Model

Dom innebär

att manipulera HTML.

Vi skulle kunna använda JS för att manipulera denna sida.

 **querySelector** med denna funktionen så får du tillbaka det första objektet/elementet som matchar selektorn.

**querySelectorAll** med denna funktion så får du tillbaka alla element/objekt som matchar selektorn i en lista.

Så här skulle syntaxen för att välja ett element eller ett attribut.

**<body>**

**<main>**

**<div>**

**<p></p>**

**<div>**

**</main>**

**<p id =”aa”></p>**

**<p class=”bb”></p>**

**<body>**

Om jag vill komma åt p-elementet i div:en i main.

**Const mainElement = document.querySelector(’main > div > p’);**

man gör alltså en variabel och använder en inbyggd funktion och skriver in namnet/identifieraren/selektorn utav det element/objekt/taggen som man vill komma åt.

om man vill komma åt ett specifikt ID så skriver man ett så här.

**Const idElement = document.getElementById(’aa’);**

Jag vill komma åt den andra p-taggen med ID: ”aa”.ll

Om man vill komma åt en tagg med en klass så använder man denna syntax:

**Const classElement = document.querySelector(’p.bb’)**

Eller:

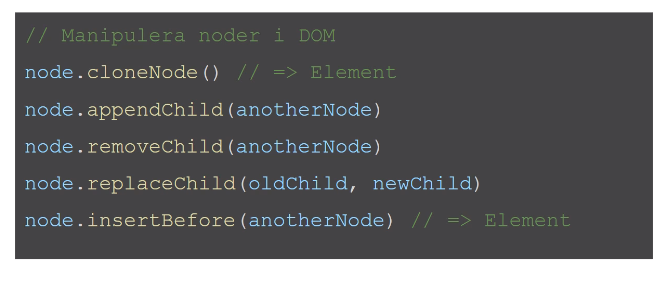
**Const classElement = document.getElementsByClassName(’bb’)**

Jag försöker komma åt p-taggen med id ”bb”.

Om du målsäter ett stort element som ”body” eller ”htlm” så appliceras alla attribut och funktioner som du skapar även på de taggar/objekt som befinner sig i den du riktade in dig på. Precis som i CSS.

Om jag hade försökt ändra på ”body” ovan så skulle allt jag gör också appliceras på ”main” osv.

### Manipulera DOM/Skapa nya HTML element.



Man kan använda sig utav JavaScript för att bygga HTML-element.

Säg att jag vill lägga till en ”section” i main

**<body>**

**<main>**

**</main>**

**</body>**

Då skriver jag i JS

**Const mainElement = document.querySletctor(’main’)** jag skapar en variabel för min container

**Const sectionElement = document.createElement(’section’);** jag skapar ett nytt element

**mainElement.appendChild(sectionElement);** jag placerar mitt skapade element i containern.

Vi kanske vill lägga till en titel och lite text i vår section.

**const title =createElement(’h2’)** skapar ett h2 element.

**const text = createElement(’p’)** skappar ett p-element

**const textbit = document.createTextNode(”hej hej hej hej”);** skapar en bit text…

**text.appendChild(’textbit’);** …och stoppar i den i p elementet vi skapade tidigare.

**sectionElement.appendChild(’title’);** till sist så placerar vi ut vår titel och vår text i sektionen.

**sectionElement.appendChild(’text’)**

resultatet skulle bli så här:

**<body>**

**<main>**

**<section>**

**<h2></h2>**

**<p>hej hej hej hej</p>**

**</section>**

**</main>**

**</body>**

man kan också skriva variabeln.

Säg att jag nu vill ta bort titeln (h2).

Och ersätta texten (p) med en section.

Dåkan jag göra så här:

**sectionElement.removeChild(’title’);** tar bort titeln

**sectionElement.replaceChild(’text’, ’sectionElement’);** ersätter texten med en section.

Resultatet skulle bli så här:

**<body>**

**<main>**

**<section>**

**<section>**

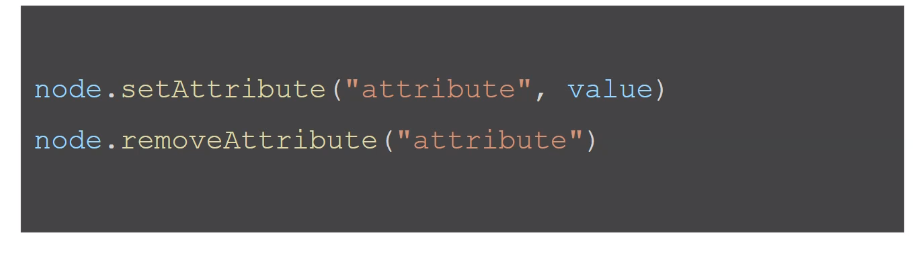
**</section>**

**</section>**

**</main>**

**</body>**

## Ändra attribut



**<body>**

**<main>**

**<section>**

**<h2></h2>**

**<p>hej hej hej hej</p>**

**</section>**

**</main>**

**</body>**

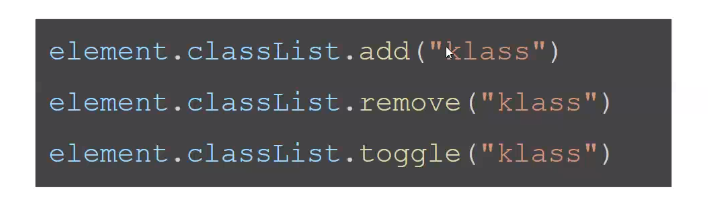
Säg att jag vill lägga på attributet på en klass attribut på <p>. Dvs jag vill lägga på ett beskrivande värde.

I JS

**Let element = querySelector(’p’);**

**Element.setAttribute(’class’, ’enKlass’)**  lägger på ett attribut utav typen ”clas” med värdet ”enKlass”.

## Ändra klasser



om man vill lägga till eller ta bort en klass från ett element så använder man

**classList.add** eller **classList.remove.** Med **classList.toggles** så kan man göra båda med en funktion. Om klassen med namnet du anger inte finns på elementet så kommer den att läggas till. Om den redan finns så tar den bort den.

## Event, att göra något när användaren gör ett input

Man vill ofta kunna göra olika saker när användaren säg, trycker på en knap.

Då använder man event.

**<button>**

**Klicka på mig**

**</button>**

Ovan har jag skapat en knap med texten ”klicka på mig” i en HTML sida.

Man kan bestämma ett beteende när knappen trycks med JS.

I JS:

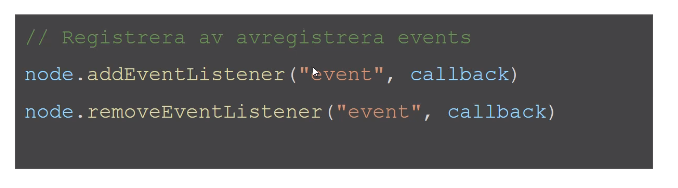
**Const btnElement = document.querySelector(’button’);** skapar en variabel för vår knap.

**btnElement .addEventListner(’click’, render);**

med klassen ”addEventListener” så kan jag övervaka vad som sker med det elementet och lägga till ett villkor och en funktion som ska köras om villkoret uppfylls.

I detta fal så övervakas knapen och om den trycks så körs funktionen ”render”.

Exempel på syntax:



Man kan också skriva så här:

**btnElement.onClick = function(){console.log(’hej)’};**

# Dag 4 2020-11-25

Om man använder sig utav

**Document.querySelectorAll()**

Denna funktion ger tillbaka en array.

Så om man vill göra något med dem så måste man hantera dem som en array, dvs specificera vilken position på det elementet som man vill göra något på eller loopa genom hela array:en.

Även om det bara var ett element man ville ha (då är de kanske inte så bra att använda).

**Documen.getElementByClassName()**

Denna funktion ger tillbaka en ”html-collection”.

Skillnaden är man behöver använda lite olika funktioner.

Ett fal när getElementByClassName kan vara utav föredel att använda är när man har ett id på en utav elementen man hämtar och man vill åt det.

**<html>**

**<button class=”knap”></button>**

**<button class=”knap”></button>**

**<button id=”knap3” class=”knap”></button>** vi vill loga denna knap utav de 3

**Let btn1 = document.getElementByClassName(’knap’)**

**Console.log(btn1[’knap3’])**

# Dag 5 2020-11-30

## API, Application Programmming Interface

API är vad man vanligtvis kallar en webtjänst. Det är en server som man kan skicka förfrågningar till och den ger då svar beroende på vad för data den har och hur den har kodats.

Ett exempel vore en server till en webbshop. Man kan skicka en förfrågan för att få ett objekt som innehåller alla varor som finns på lager och deras pris etc. Man kan även skicka data till dem tex ändra lagersaldo ifall en produkt köps.

All Kommunikation sker över http.

När man skickar en förfrågan så gör man det via en URL-sökväg.

### http Verb

Man ska även skicka med ett ”**http-verb**”. Detta är ett kommando som säger till servern vad du vill göra. Om du vill hämta data, ändra data, ta bort data etc. Beroende på vad du skriver och skickar så svarar API:n på ett visst sätt.

Exempel:

**GET** Hämta data, det är denna metod vi ska arbeta med och kunna.

**POST** Skappa data på API:ns sida.

**PUT** Ersätta ett vist data på API:ns sida.

**PATCH** Ändra ett data på API:ns sida.

**DELETE** Ta bort ett data på API:ns sida.

Det finns fler men de är inte relevanta nu.

### http Parameters

Du kan också skicka med parameter som ytterligare specificerar vad du vill åt. Tex om du vill åt flera objekt eller om du vill ha ett slumpat objekt.

Vanligtvis så skickar man dessa parametrar i <url:en> men läs dokumentationen på API:n för exakta instruktioner.

### HTTP Response

När du gjort din request (bet API:n om data) så får du ett svar. Svar består utav 3 delar

#### Status code

En sifferkod som beskriver vad som har hänt/hur det har gått. 200-tal om allt har gått ok. 400-tal om det blev något fel från din sida. 500-tal om det blev fel från serverns sida.

#### Headers

Meta data, inloggningar etc.

#### Response body

Den data som du bad om. Formaterat på ett visst sätt. Vanligt är JSON.

## JSON, Java Script Object Notation

JSON är ett standardformat som beskriver hur man ska formatera data. Alltså hur data man får tillbaka kommer att se ut.

JSON beskriver att data ska skickas som Objekt. I objektet ska Nyckeln vara en sträng. Värdet kan vara vad det vill (nummber, String etc).

Så om du gör en fetch till en API som använder sig utav JSON så kommer data att se ut något så här:

Nyckel: data

**Let data = [**

**{**

**”namn”: ”Gustav”,**

**”ålder”: 24,**

**”land”: ”Sverige”**

**},**

**{**

**”namn”: ”kicki”,**

**”ålder”: 25,**

**”land”: sverige**

**}**

**]**Som du ser ovan så kommer data att vara i ett objekt där nycklarna är en sträng och värdena kan variera i data typ. Man kan också formatera data som json själv efter att man fåt den.

## Fetch

Fetch är en inbyggd metod för att anropa till API:er. Den gör en http-request åt dig och returnerar ett **Promise** objekt.

### Hur man gör en Fetch/Syntax för Fetch

**Fetch(’url’)** skriv metoden fetch. Du skickar in den <url:en> du vill göra en request från i metoden.

**.then(response => response.json())** här säger vi till att datan ska sparas i json format.

**.then(data => {** Här bestämer du vad som ska göras med den data du får tillbaka.

**console.log(data)** I detta fal så logar jag datan så att jag kan inspektera den och sparar den som en variabel så att jag kan använda den i fortsättningen.

**const dataSave = data;**

**})**

**.catch(error => console.log(error));** Om något går fel, text inget internet så loggar vi felkoden och dess info.

### .then

Det är med funktionen ”**.then**” som man kontrollerar vad som ska göras med det man får tillbaka från API:n.

# Dag 6 2020-12-03

## Promise

Promise är ett objekt som kan ändra sitt värde asynkront. Dvs ett värde sätts först och koden fortsätter vidare men objektet håller koll på sig själv och kan komma att ändra på sig när nya parametrar kommer in.

När vi gör en förfrågan så får vi tillbaka ett promise som ändrar på sig när man får tillbaka data.

Om man inte skulle använda sig utav prommise så skulle man behöva vänta på svar från servern innan koden skulle kunna fortsätta. Och när man skickar en förfrågan till en API så måste ju både förfrågan och svaret hitta sin väg över internet. Detta kan ta vis tid.

Ett alternativ är att vänta på att man får ett svar men det är inte särskilt effektivt.

## Fetch

Vad som händer istället är att när man använder sig utav fetch metoden är att den ger ett omedelbart svar i form utav ett promise. Efter att man har gjort sin förfrågan så fortsätter koden att köras medan vi väntar på svar och vårt promise håller koll på den förfrågan.

Ett promise är ett objekt med tre olika värden:

**Pending** |du har inte fått något svar.

**Fulfilled** |du har fått ett svar

**Rejected** |ditt försök att hämta data harmisslyckats.

De viktiga här är att man kan kontrollera vad som ska hända beroende på vilket svar man får. Med funktionen **.then** så kontrollerar man vad som ska hända om förfrågan till API:n lyckades. Alltså att vi fått den data vi bad om. Med funktionen **.catch** så kan du kontreolera vad som ska göras om förfrågan misslyckades.

## Async/Await

En annan syntax för att arbeta med ”prmoises”.

En egenskap med Async/Await är att man kan be koden att pausa när man gjort en förfrågan. Alltså om det är en vis data som är kritisk och du inte vill göra något annat innan du får den data att arbeta med så kan man pausa kod flödet.

Async skrivs som funktioner. Vill man göra en async/await function så använder man sig utav nyckel ordet **async** framför funktionen.

Exempel på Aync syntax:

**Async function getData(url, callback){** skapar en async funktion som ska hämta data. Indata är: <URL:en> vi sk göra en förfrågan till och en funktion som vi ska göra en callback till.

**Const response = await fetch(url);** vi spara svaret som en variabel. Notera ”await”. Till dess att vi har fått ett svar så sparas inget värde och vi gör inget annat.

**Const data = response.json();** När vi väl har fått ett svar så formaterar vi det som ett json objekt.

**Callback(data);** Och vi skickar data till vår funktion.

**}**

# Dag 7 20-12-09, Local storage. Och Axios

## Axios

Axios är ett alternativ till Fetch. Dvs ett sätt att hämta data från ett API.

Axios är dock inte inbyggt i JavaScript utan man måste skapa en referens i HTML.

I head skriv:

**<head>**

**<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>**

**</head>**

Denna scripttagg kan du hitta på github om du googlar axios js. Längst ned på denna sida.

<https://github.com/axios/axios>

synatx ser ut så här:

**async function getUser(){**

**try{**

**const response = await axios.get(’url’);**  url= API du vill hämta data ifrån.

**Console.log(response);**

**Catch (error){**

**Console.log(error);**

**}**

**}**

Anat än att syntaxen så funkar axios likadant som fetch. Dvs du får tillbaka ett objekt med olika ”properties” som är upp till dig att behandla. En skillnad är att axios ger data i formatet json automatiskt.

## Local storage

## 

Man kan använda JavaScript för att se till att data sparas lokalt på enhet (i webbläsaren) som kör hemsidan och des script.

Detta kan vara användbart när man ska ha flera sidor på sin hemsida att växla mellan och man har data som ska användas på alla sidor. Du sliper då göra fler anrop eller skapa nya variabler/objekt.

Om man sparar data lokalt så kan man sedan referera till den och bygga funktioner kring den.

Detta kallas lokal storage.

För att spara något lokalt så gör man så här:

**LocalStorage.setItem(’myDog’, ’dog’);**

Allt som du sparar lokalt sparas i ett specielt objekt. I koden ovan så anger jag nyckeln följt utav värdet. Alltså så sparade jag en property i local storage objektet med nyckeln ”my dog” och värdet ”dog”.

för att se vad som finns sparat lokalt så använder man sig utav webbläsarens utvecklarverktyg under fliken **application**.

För att sedan hämta ett värde som är sparat lokalt så skriver man:

**Let hund = localStorage.getItem(’myDog’);** jag sparar värdet som en variabel.

**Console.log(hund);** och skriver ut det.

**”dog”**

## Session storage

Ett alternativ till local storage är session storage. Den gör exakt samma sak men datan raderas när du stänger fliken. Datan har alltså en kortare livslängd.

Syntax:

**SessionStorage.setItem()**

**Let värde = sessionStorage.getItem();**

**sessionStorage.clearItem(’dog’)** tar bort ett värde

**sessionStorage.clear();** tar bort allt i session storage