**Laborationsrapport**

**Moment 4 / Ecmascript & Typescript**

*DT173G, Webbutveckling III*

**Författare: Felicia Gillstörm,** [**fegi2000@student.miun.se**](mailto:fegi2000@student.miun.se) **Termin, år: HT, 2021**



# Sammanfattning

Beskriv kortfattat vad uppgiften går ut på.

# Innehållsförteckning

[1](#_Toc83040795)

[1 Sammanfattning 2](#_Toc83040796)

[Beskriv kortfattat vad uppgiften går ut på. 2](#_Toc83040797)

[2 Innehållsförteckning 3](#_Toc83040798)

[3 Frågor 4](#_Toc83040799)

[3.1 Förklara begreppet "Ecmascript", samt dess historia för JavaScript. Gå igenom vilka olika versioner som kommit genom åren, och nämna några av de större uppdateringarna som kommit med vardera version. 4](#_Toc83040800)

[3.2 Förklara följande tekniker, med kod-exempel för vardera: 4](#_Toc83040801)

[3.3 Förklara Fetch API och Promises 4](#_Toc83040802)

[3.4 För bakåt-kompabilitet - hur kan vi konvertera senare versioner av Ecmascript till något som fungerar i "alla" webbläsare med Gulp (alternativt Webpack)? Uppdatera ditt Gulp- eller Webpack-projekt från tidigare laboration(er) och publicera denna till Github eller annan webbtjänst för Git. Lägg in kommandot för att klona ditt repo till lokal dator som en del av svaret på denna fråga. 4](#_Toc83040803)

[4 Analys av Typescript 5](#_Toc83040804)

[5 Slutsatser 6](#_Toc83040805)

[6 Källförteckning 7](#_Toc83040806)

# Frågor

## Förklara begreppet "Ecmascript", samt dess historia för JavaScript. Gå igenom vilka olika versioner som kommit genom åren, och nämna några av de större uppdateringarna som kommit med vardera version.

Ecmascript är ett scriptspråk som är standardiserat av Ecma International, förkortningen Ecma står för European computer manufacturer association. Det utvecklades första av Brendan Eich på Netscape och släpptes 1995 men hette då ifrån början Mocha men döptes sedan om först till LiveScript och slutligen till JavaScript. När man insåg att JavaScript skulle användas till andra webbläsare än bara Netscape så skapades en så kallad ecmaskript-standard som släpptes 1997 och var till för alla webbläsare. Sedan de skapades så har det kommit en hel del uppdaterade versioner, här nedan redovisas dessa:

* ES1 (1997)
* ES2 (1998) Ändringar gjordes för att fungera med ISO/IEC 16262 internationella standard.
* ES3 (1999) Lade till bättre string-hantering, nya control statements, bättre definition för errors och formaterades för numeriskt output till exempel.
* ES4 (2003) Slopades hellt och publicerades aldrig på grund av politiska skillnader när de gällde språkets komplexitet. Många av de funktioner som var tänkt att användas försvann helt men en del återupplivades i den sjätte upplagan.
* ES5 (2009) Lade till ”strict mode” ett subset som var avsedd att ge en mer grundlig felkontrollering. Det tydliggörs även den del oklarheter som fanns i den tredje upplagan. Nya funktioner lades till såsom getters setters och även support för JSON.
* ES5.1 (2011) Standarden vart då helt i linja med den tredje upplagens intternationella standard.
* ES6 (2015) Döptes sedermera om till ECMAscript2015. I denna uppdatering sån lades det till ny syntax för mer komplicerade applikationer som inkluderade klasser. Det lades även till iterationer, loopar, let och const. Även arrow-funktioner och promises tillkom med denna version.
* ECMAscript2016 (2016) Här lades block-scoping för variabler och funktioner. Det lades även till array.prototype.include funktionen. Även ett enklare sätt att arbeta med promises tillkom med hjälp av async/await.
* ECMAscript2017 (2017) Lade till funktioner för enklare manipulation av objekt.
* ECMAscript2018 (2018) Nya features såsom spread operatorer, asynkronisk iteration samt tillägg till RegExp.
* ECMAscript2019 (2019) Lade till möjligheten till includes men var inte limiterad till array.prototype.flat. Array.sort var med denna uppdatering även garanterad att vara stabil, där menas att element med samma sorteringsförekomst kommer att visas i samma ordning i den sorterade arrayen.
* ECMAscript2020 (2020) Här tillkommer BigInt funktionen.
* ECMAscript2021 (2021) Är den senast versionen, här introduceras vi till replaceAll metoden för strings bland annat.

Mer går att nämna om alla dessa versioner men det skulle kunna ta en evighet så vi går istället vidare och förklarar lite av de nyare funktioner och visar upp dessa med olika kod-exempel.

## Förklara följande tekniker, med kod-exempel för vardera:

* **Classes**  
  Är ett sätt att objektorientera i javaScript, det inkluderar traditionelle saker som till exempel this, extend och constructor. Ett exempel på att använda classes och enkelt lägga till mer i classes är att skriva:  
    
  class Cat {  
  constructor (name, breed) {  
  this.name = name;  
  this.breed = breed;  
   }  
  }  
   /\*Skapar en ny instans till klassen Cat\*/  
  let c = new Cat(”Tusse”, ”Sphynx”);
* **High-order functions**Är funktioner som använder sig av andra funktioner, antingen igenom att returnera eller använda dessa funktioners argument. Här nedan visas ett sådant exempel:  
    
  function greaterThan(b) {  
  return a => a > b; //detta är en arrowfunktion som vi kommer gå igenom sen  
  }  
    
  let greaterThan3 = greaterThan(3);  
    
  console.log(greaterThan3(4)); // => true returneras.

* **High order array methods - forEach, map och filter**  
  Higher order arrray methods är metoder som kan användas i arrayer för att få fram den väsentliga informationen av det som är lagrat där.  
    
  Om vi har en array med frukter till exempel och vi vill skriva ut alla frukter som finns i den arrayen då kan man använda sig av forEach som loopar igenom arrayen och för varje loop så skrivs nästa objekt ut i arrayen.  
    
  let fruits = [’banana’, ’kumquat’, ’pineapple’, ’orange’];  
    
  fruit.forEach (element => console.log(element));

Om vi har en array av objekt som representerar script men vi vill att den skall vara lättare att inspektera, då kan man till exempel använda sig av map. Map omvandlar en array genom att tillämpa en funktion till alla dess element och därigenom bygga en ny array ifrån de returnerade värdena.  
Den nya arrayen har samma längd som original arrayen men innehållet har mappats till en ny form.  
  
let numberArray = [2, 3, 4, 5, 35]

let numberArray = numberArray.map(numberItem => {

return numberItem \* 2

}

/\*arrayen kommer efter detta att innehålla [4 ,6 ,8 ,10 ,70 ]\*/  
  
  
Genom att använda filter så kan du välja ut just de delar av en array som passerar det krävande momentet utan att ta bort element ifrån den redan existerande arrayen.  
  
let ages = [15, 32, 17, 41, 19, 16];  
  
ages.filter(checkAdult) // Returns [ 32, 41, 19]

function checkAdult(age) {

return age >= 18;

}

* **Spread operators ("spreads")**  
  Spread operators används när vi vill extrahera innehåll ifrån exempelvis ett objekt eller en array. När man skriver ett spread så används tre punkter ”…” Det används tillexempel för att omvandla en array till en textsträng och vise versa.  
    
  const cityArr = ”sundsvall”;  
  let characters =[…cityArr];  
    
  /\*detta blir en array som innehåller [s, u, n, d, s, v, a, l ,l ]\*/  
    
  Områderna är många för spread, man kan även använda det för att plocka ut det lägsta talet i en array eller använda det som argument i andra funktioner.
* **Destructuring**  
  Destructring används för att extrahera värden ifrån ett objekt eller en array.  
    
  let a, b, theRest;  
  [a, b] = [1, 2] //gör att a tilldelas värdet 1 och b tilldelas 2  
    
  /\*skrivs flera värden så hamnar de i theRest\*/  
    
  [a, b, …theRest = [1, 2, 3, 4, 5] //a =1, b=2 theRest = [3, 4, 5]
* **Arrow Functions**  
  Är det tredje sättet som man kan deklarera funktion på. Man använder sig av =>  
  Här nedan visas en enklar kod för hur man kan använda arrow functions.  
    
  let sayHello = () => {  
   console.log(”Hello there!”);  
  }  
    
  innehåller arrow funktionen bara ett argument så behövs inte paranteserna och innehåller funktionen bara ett statement så behövs inte måsvingarna heller, detta gör att koden kan kortas ned och förenklas rejält.

## Förklara Fetch API och Promises

Fetch API är en enklare och senare variant än det tidigare AJAX med XMLHttpRequest där man använder sig av lite annan syntax. För att göra ett anrop med Fetch så skrivs först;

fetch(url) //här anropas det med hjälp av fetch och skickar adressen som en url  
  
.then(function() {  
  
//hanterar datan som kommer med anropet  
  
})  
  
.catch(funtion() {  
  
//plockar upp ifall det kommer några felmeddelanden  
  
});  
  
  
Promises ger oss möjligheten att få en lite mer läsbar kod med bättre hantering av asynkrona händelser med bättre programflöde samt bättre felhantering.  
Promises finns i fyra olika states, de är: resolved/fullfilles, rejected, pending och settled.  
Detta gör att om vi har en funktion som skall göra flera saker så kan den göra det efter att varje del har blivit ”resolved” man kan även så information ifall något blivit ”rejected” alltså när ett krav inte blivit resolved.

## För bakåt-kompabilitet - hur kan vi konvertera senare versioner av Ecmascript till något som fungerar i "alla" webbläsare med Gulp (alternativt Webpack)? Uppdatera ditt Gulp- eller Webpack-projekt från tidigare laboration(er) och publicera denna till Github eller annan webbtjänst för Git. Lägg in kommandot för att klona ditt repo till lokal dator som en del av svaret på denna fråga.

# Analys av Typescript

# Slutsatser

Vilka slutsatser har du dragit av uppgiften?

# Källförteckning

Här följer exempel på hur en källförteckning kan utformas enligt Vancouver-systemet. Den är automatiserad enligt metoden numrerad lista och korsreferenser. Radera denna text, samt ersätt källorna med dina egna.

1. Monash University, ”Citing, Vancouver Style” Publicerad 2006-04-13. Hämtad 2012-01-25 (Exempel på referens till webbsida)
2. Svenska Datatermgruppen, ”Information om datatermer”,  
   http://www.nada.kth.se/dataterm/   
   Publicerad 1998-08-20. Hämtad 2005-04-11. (Exempel på referens till webbsida)