Laborationsrapport

Moment 4 / Ecmascript & TypeScript
DT173G, Webbutveckling III

Författare: Elias Eriksson, eler2006@student.miun.se
Termin, år: HT, 2021

Mittuniversitetet
MID SWEDEN UNIVERSITY

Campus Härnösand Universitetsbacken 1, SE-871 88. **Campus Sundsvall** Holmgatan 10, SE-851 70 Sundsvall.
Campus Östersund Kunskapens väg 8, SE-831 25 Östersund.
Phone: +46 (0)771 97 50 00, Fax: +46 (0)771 97 50 01.

Sammanfattning

Uppgiften går ut på att få en insikt i ECMAScript historia, vilka funktioner som har lagts till i ECMAScript för att göra det lättare att skriva JavaScript. Hur vi kan utnyttja de nya funktionerna och även låta användare med webbläsare från 1990 använda webbplatsen. Samt insikt i vad TypeScript är och hur det kan hjälpa oss utvecklare utveckla lättare.

Innehållsförteckning

1 Frågor

1.1 Fråga 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.

ECMAScript är en standard som introducerades 1997 [1] 2 år efter att Netscape släppte JavaScript [2]. Standarden introducerades så att alla webbläsare på marknaden skulle skulle vara kompatibla med alla webbsidor. Även fast en standard fanns så valde Microsoft att sluta samarbeta med ECMA när Internet Explorer marknadsandel växte runt år 2000. Mellan 2005-2008 gav sig Mozilla och Google in på webbläsarmarknaden. 2008 gick dessa tre parter ihop och kom överens om att fortsätta med en standard vilket. Första steget vart ECMAScript 5 år 2009.[2]

Urval av funktioner som kommit med ES versionerna.

- ES3: regex och try / catch error hantering. [1]
- ES5: strict mode, JSON support. [1]
- ES6: Classer, moduler, const och let, arrow funktioner, promises,[1] tempalte literals.
- ES7: Destructuring, async / await. [1]
- ES8: object manipulering med Object.values, Object.entries och Object.getOwnPropertyDescriptors. [1]
- ES9: spread operatorn funkar med object, rest parametrar. [1]
- ES11: En hel del array metoder bland annat array flat som skapar 1d arrayer av nd arrayer. ?? och optiona chaining med ?. [1]
- ES12: replaceAll metod till strängar. [1]

ES4 skulle ha kommit med mycket av de funktionerna vi ser i ES6-ES7 och även mer som valbar typning. Men ES4 skrotades då den utvecklades under perioden då Microsoft slutade samarbeta med ECMA. [1]

1.2 Fråga 2

Förklara följande tekniker, med kod-exempel för vardera:

- **Classes**

Klasser är en teknik som används för att kapsylera data och kan beskrivas som en mall för att skapa objekt. [3]

```
class Cat {
    constructor(name, breed, age) {
        this.name = name;
        this.breed = breed;
        this.age = age;
    }

    log = () => {
        return `${this.constructor.name}(${this.name}, ${this.breed}, ${this.age})`
    }

    attributes = () => {
        return [this.name, this.breed, this.age];
    }
}

const cats = [
    new Cat( name: "Sotis",  breed: "bondkatt",  age: 3),
    new Cat( name: "Findus",  breed: "bondkatt",  age: 4)
]
```

- **High-order functions**

En higher-order funktion är en funktion som tar en funktion som en parameter och /eller returnerar en funktion. [4] Det går att skriva egna och JavaScript har även några inbyggda som forEach och map (array metoder). Exempel på en egenskriven timer funktion.

```
const timer = (func, ...args) => {
    return async () => {
        console.log(`starting to time function ${func.name}...`);
        const start  = new Date();
        let result = await func(...args);
        const end = new Date();
        console.log(`Function ${func.name} took ${end.getTime() - start.getTime()}ms to complete.`);
        return result;
    }
}

const timed = async (second) => {
    console.log(`Waiting ${second}s.`);
    await new Promise( executor: resolve => setTimeout(resolve,  timeout: second * 1000));
    console.log(`Finished waiting ${second}s.`);
}


timer(timed,  args: 4)();
timer(timed,  args: 3)();
```

- **High order array methods - forEach, map och filter**

forEach tar en funktion som en obligatorisk parameter som kommer kallas en gång med varje element från arrayen. [5]

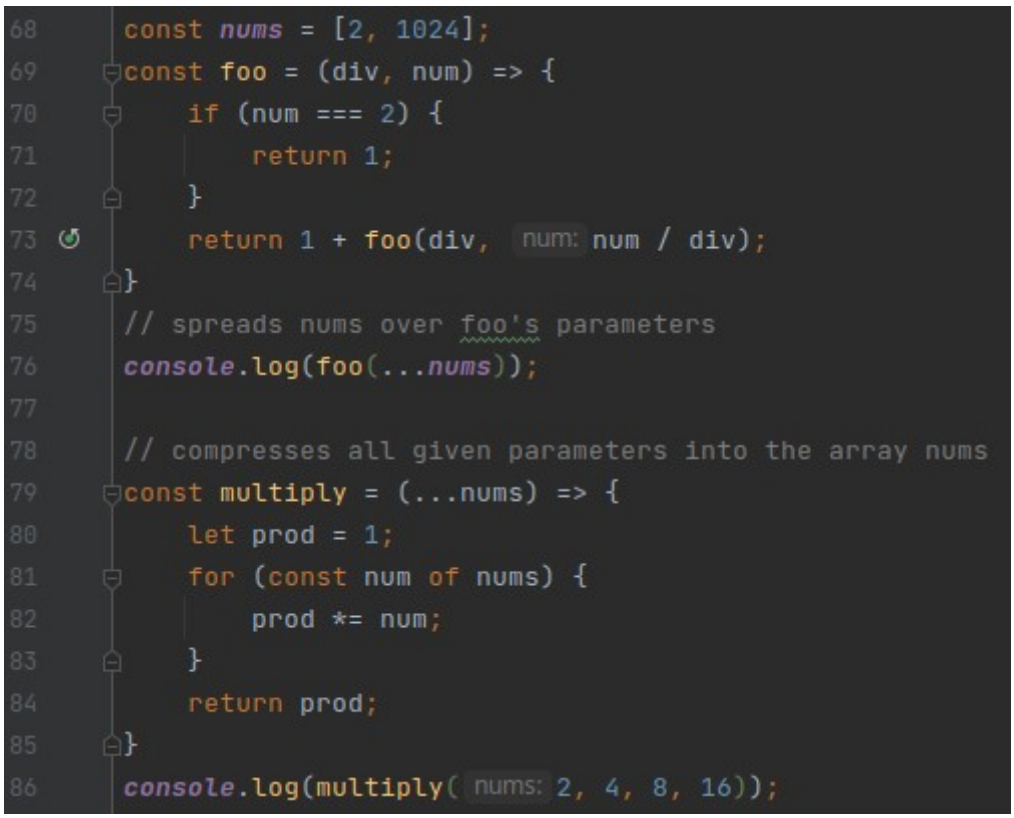
map tar en funktion som en obligatorisk parameter som kommer att kallas en gång med varje element från arrayen och en ny array kommer skapas från funktionens returvärden. [6]

filter tar en funktion som en obligatorisk parameter som kommer kallas en gång med varje element från arrayen och en ny array kommer att skapas från de element som get ett ”truthy” returvärde från funktionen. [7]

```
48  console.log("Cats")
49  cats.forEach((cat :Cat ) => console.log(cat.log()));
50
51  // map example
52  const olderCats = cats.map(cat => {
53      let [name, breed, age] = cat.attributes();
54      return new Cat(name, breed,  age: age+2);
55  });
56  console.log("Older Cats");
57  olderCats.forEach(cat => console.log(cat.log()));
58
59  // filter example
60  const oldCats = cats.filter(cat => {
61      return cat.age > 3;
62  });
63  console.log("Old Cats");
64  oldCats.forEach(cat => console.log(cat.log()))
```

- **Spread operators ("spreads")**

Sprid operatorn används antingen när en array variabel innehåller värden som ska spridas över flera funktionsparametrar[8] eller när en funktion deklareras och parametrar ska lagras som en array. [9]



- **Destructuring**

Destrukturering gör det möjligt att packa upp värden från en array eller objekt och benämna värdera från arrayen/objektet med egna variabler. Det går även att använda … operatorn som en rest variabel vid array destrukturering. [10]

```
// array destructuring
const add = (a, b) => {
    return [a + b, a, b];
}

let [sum, left, right] = add( a: 4,  b: 8);
console.log(sum , left, right);

let added;
[sum, ...added] = add(sum,  b: sum * 2);
console.log(sum, added);

// object destructuring
const logCat = ({name, breed, age}) => {
    console.log(`The cat ${name} of breed ${breed} is ${age} years old.`);
}

cats.forEach(logCat);
```

- **Arrow Functions**

Pil funktioner är ett annat sätt att skriva funktioner i Javascript. Det anses vara ett mer kompakt sätt att skriva funktioner. Pil funktioner skiljer sig dock lite från vanliga funktioner då pil funktioner bland annat inte ändrar på nyckelordet this. [11]

```
class Foo {
    constructor(a) {
        this.a = a;
        window.addEventListener( type: "load",  listener: () => {
            // this.a does refer to Foo.a
            console.log(`arrow function ${this.a}`);
        });

        window.addEventListener( type: "load",  listener: function () {
            /// this.a does NOT refer to Foo.a
            console.log(`regular function ${this.a}`);
        });
    }
}
```

1.3 Fråga 3

Förklara Fetch API och Promises

Promise är en typ av objekt som kan ha tre olika stadier: resolved / fullfilled, rejected eller pending. Ett Löfte kommer vara i ett pending stadie när det skapas. För att skapa ett promise ges en funktion som tar två valfria parametrar som är 2 funktioner. Typiskt sätt benämnda resolve respektive reject. Ett promise objekt ändrar inte sitt state från pending till resolved eller rejected fören någon av dessa två funktioner har blivit kallade. Både reject och resolve tar en parameter och fungerar ungefär som return / throw fast för promises. Om ett promise lyckas går det att få ut resolve funktionens parameter genom att antingen vänta på promise objektet med await alternativt så används promise objektets then() metod. then() metoden tar en funktion som parameter (alternativt 2 där error hantering är den andra men det brukar oftast hanteras med .catch) som i sin tur tar en parameter. Parameterns värde kommer att vara samma som den parametern som gavs i resolve funktionen. Det går att kjedja flera .then metoder efter varandra så länge den tidigare .then metoden retunerade ett promise. Det fungerar på samma sätt för rejected men typiskt sätt så är det .catch metoden används för att fånga det som gavs ner i reject funktionen.[12]

```
131  // creates a promise
132  const p = new Promise(((resolve, reject) => {
133      let number = Math.random();
134      if (number > 0.5) {
135          resolve(number);
136      }
137      reject(number);
138  }));
139
140  // async / await, error handling with javascript try / catch
141  // set timeout is used to get into an asynchronous scope so await can be used.
142  setTimeout( handler: async () => {
143      try {
144          let number = await p;
145          console.log(`(A)The generated number was larger than 0.5. ${number}`);
146      } catch (e) {
147          console.log(`(A)The generated number was smaller or equal to 0.5 ${e}`);
148      }
149  });
150
151  // .then() .catch()
152  // (this is the same promise as above and will not regenerate a new number)
153  p.then(number => {
154      console.log(`(T)The generated number was larger than 0.5. ${number}`);
155  }).catch(e => {
156      console.log(`(T)The generated number was smaller or equal to 0.5 ${e}`);
157  });
```

Fetch api är ett annat sätt att göra webb anrop än XMLHttpRequest. Fetch bygger på Promise och att kedja sina anrop med metoden .then eller använda async / await. [13]

```
152  // async / await
153  const request = async () => {
154      const url = "https://djangorest.eliaseriksson.eu/";
155      let response;
156      try {
157          response = await fetch(url);
158      } catch (e) {
159          console.log(e);
160          throw e;
161      }
162      let json = await response.json();
163      console.log(json);
164  }
165  request();
166
167  //.then / .catch
168  const url = "https://djangorest.eliaseriksson.eu/";
169  fetch(url).then(response => {
170      return response.json();
171  }).then(data => {
172      console.log(data);
173  }).catch(e => {
174      console.log(e);
175      throw e;
176  });
177
```

1.4 Fråga 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.

Ett sätt är att använda en transpiler som omvandlar senare versioner av Javascript till tidigare versioner av Javascript. Ett exempel på en transpiler är Babel som stöder att omvandla ES6+ kod till tidigare versioner. Gulp har stöd för Babel via paketet gulp-babel.[14][15]

repo: https://github.com/EliasEriksson/miun-web3-mom4

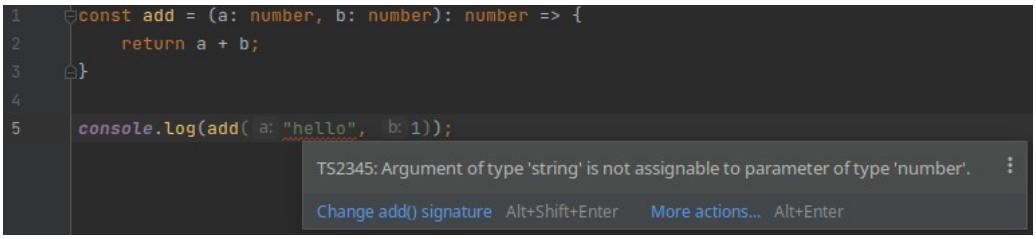
1.5 Fråga 5p

Gör en analys av Typescript. Analysen bör vara i storleksordningen en till två A4-sidor. Använd kod-exempel och illustrationer.

TypeScript började utvecklades av Microsoft under 2010 talen när ECMAScript version 5 va standard versionen av JavaScript. Målet med att utveckla TypeScript var att göra det lättare att utveckla med JavaScript då det ansågs vara en utmanande att bygga en större applikation med JavaScript. [16]

Eftersom att TypeScript är ett super set av JavaScript/ECMAScript betyder det att TypeScript kommer med funktioner som inte finns med i den standard som webbläsarna följer. Det är alltså inte möjligt att köra TypeScript i en webbläsare utan att först kompilera TypeScript till JavaScript. Detta kan göras genom att installera npm paketet typescript [18]. Om det snarare är av intresse att köra koden på server sidan går det att kompilera ner till JS och sedan köras med Node.js eller så kan TypeScript köras direkt med Deno [19].

Det som idag skiljer TypeScript från ECMAScript är att TypeScript har möjligheten till statisk typning [16]. Statisk typning gör det möjligt för (i detta fall) TypeScripts compiler att fånga många fel från programmeraren som skulle kunna leda till errors vid produktion. Om en funktion förväntar sig att en parameter är numerisk och programmeraren ger en string kommer TypeScripts compiler att reagera och ge fel även fast det är tillåtet i vanlig ECMAScript.



När ett projekt växer och blir större blir det svårare att veta vilka parametrar som har vilken typ då flera olika utvecklare kanske jobbar på samma kod bas och att då ha möjlighet till statisk typning gör det mycket lättare att hantera projektet. [17]

Eftersom att utvecklingsmiljöer kan dra nytta utav statisk typning genom att ge feedback på vilken typ som ska vara i vilken parameter och eftersom miljön känner igen typen på en parameter så känner den även igen metoder kopplade till typen så blir det mycket lättare att utveckla. [17]

Om vi som utvecklare till exempel ska skriva en funktion som adderar 2 nummer med varandra och resultatet sedan loggas så skulle våran JavaScript kunna se ut såhär:

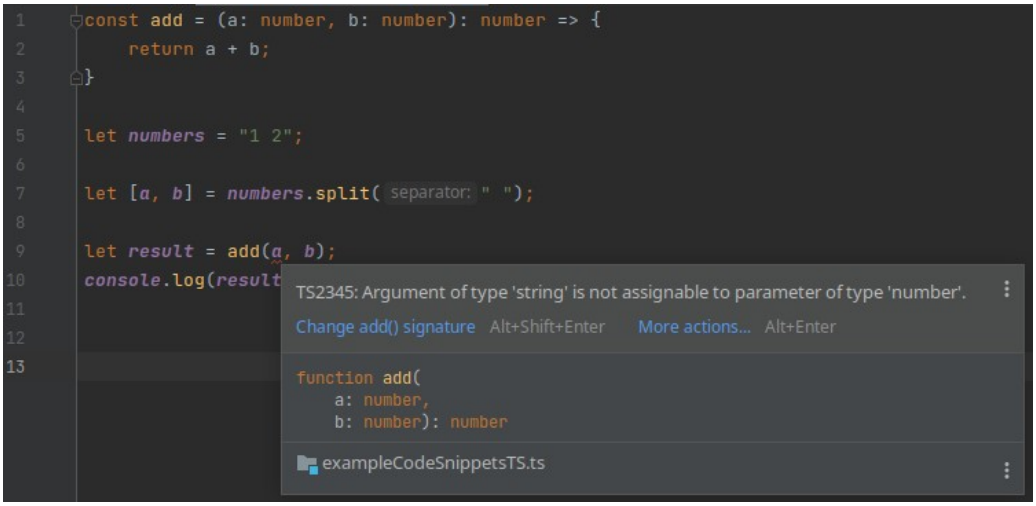
```
const add = (a, b) => {
    return a + b;
}

let numbers = "1 2";

let [a, b] = numbers.split( separator: " ");

let result = add(a, b);
console.log(result);
```

Resultatet som vi förväntar oss efter denna körning är 3 då 1 + 2 är 3 men vid körning av denna kod så blir resultatet: 12. Eftersom att numren i text strängen aldrig vart omvandlade till numer typ. Om vi med TypeScript skriver ut att add funktionens parametrar a och b är av typen nummer kommer TypeScript compilern att fånga upp detta som en error.



Beroende på vad som görs med resultatet sen kanske ingen error någonsin skull uppstå. Om resultatet skrevs till DOM istället för att konsol loggas så hade felet kanske inte hittats fören koden låg ute i produktion och en användare ser att 1 + 2 inte blir 3 utan 12 istället.

2 Slutsatser

Sen 2015 har det kommit mycket väldigt användbara funktioner som underlättar utvecklingen. Bland annat promises, fetch API och async / await har gjort det lättare att göra webb anrop jämfört med XMLHttpRequest. Det har även kommit ut verktyg som Babel som gjort det möjligt att skriva kod med nya features som kan transpileras ner till en äldre version av ECMAScript så att flera användare kan köra koden i sina webbläsare. TypeScript finns också som ett extra verktyg vi utvecklare kan använda för att vi ska göra mindre fel under utvecklingen samt att vi får mer hjälp från våran utvecklingsmiljöer.

3 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] Wikipedia ”ECMAScript” https://en.wikipedia.org/wiki/ECMAScript

[2] Wikipedia ”JavaScript” https://en.wikipedia.org/wiki/JavaScript

[3] MDN ”Classes” https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes?retiredLocale=sv-SE

[4] Codecademy ”Learn JavaScript: Iterators” https://www.codecademy.com/learn/game-dev-learn-javascript-higher-order-functions-and-iterators/modules/game-dev-learn-javascript-iterators/cheatsheet

[5] MDN ”Array.prototype.forEach()” https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach

[6] MDN ”Array.prototype.map()” https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map

[7] MDN ”Array.prototype.filter()” https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter

[8] MDN ”Spread syntax” https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax

[9] MDN ”Rest parameters” https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/rest_parameters

[10] MDN ”Destructuring assignment” https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment

[11] MDN ”Arrow function expressions” https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

[12] MDN ”Promise” https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

[13] MDN ”Fetch API” https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

[14] Babel ”What is Babel?” https://babeljs.io/docs/en/

[15] npm ”gulp-babel” https://www.npmjs.com/package/gulp-babel

[16] Wikipedia ”TypeScript” https://en.wikipedia.org/wiki/TypeScript

[17] Dropbox ”Our journey to type checking 4 million lines of Python” https://dropbox.tech/application/our-journey-to-type-checking-4-million-lines-of-python

[18] Mattias Dahlgren ”Ecmascript & Typescript” https://play.miun.se/media/Ecmascript+%26+Typescript/0_gvp18j8h

[19] Deno ”Deno” https://deno.land/