

COURSE 2

Belajar Dasar JavaScript (Sisi Frontend + Backend)

Material Details

01 Hello JavaScript

02 Fungsi, Objek & Event

03 String & Number Methods

04 Dunia Array!

05 JavaScript Condition

06 Loop you so much

07 JavaScript DOM

08 ES6

Modul Sekolah Fullstack Cilsy

Hak Cipta © 2020 PT. Cilsy Fiolution Indonesia

Hak Cipta dilindungi Undang-Undang. Dilarang memperbanyak atau memindahkan sebagian atau seluruh isi buku ini dalam bentuk apapun, baik secara elektronis maupun mekanis, termasuk mecopy, merekam atau dengan sistem penyimpanan lainnya, tanpa izin tertulis dari Penulis dan Penerbit.

Penulis : Muhammad Lukman Hakim
Editor : Muhammad Fakhri Abdillah, Iqbal Ilman Firdaus

Penerbit : PT. Cilsy Fiolution Indonesia
Web Site : <https://cilsyfiolution.com> , <https://sekolahfullstack.com>

Sanksi Pelanggaran Pasal 113 Undang-undang Nomor 28 Tahun 2014 tentang Hak Cipta

1. Setiap orang yang dengan tanpa hak melakukan pelanggaran hak ekonomi sebagaimana dimaksud dalam pasal 9 ayat (1) huruf i untuk penggunaan secara komersial dipidana dengan pidana penjara paling lama 1 (satu) tahun dan atau pidana denda paling banyak Rp100.000.000,00 (seratus juta rupiah).
2. Setiap orang yang dengan tanpa hak dan atau tanpa izin pencipta atau pemegang hak cipta melakukan pelanggaran hak ekonomi pencipta sebagaimana dimaksud dalam pasal 9 ayat (1) huruf c, huruf d, huruf f, dan atau huruf h, untuk penggunaan secara komersial dipidana dengan pidana penjara paling lama 3 (tiga) tahun dan atau pidana denda paling banyak Rp500.000.000,00 (lima ratus juta rupiah)
3. Setiap orang yang dengan tanpa hak dan atau tanpa izin pencipta atau pemegang hak melakukan pelanggaran hak ekonomi pencipta sebagaimana dimaksud dalam pasal 9 ayat (1) huruf a, huruf b, huruf e, dan atau huruf g, untuk penggunaan secara komersial dipidana dengan pidana penjara paling lama 4 (empat) tahun dan atau pidana denda paling banyak Rp1.000.000.000,00 (satu miliar rupiah)
4. Setiap orang yang memenuhi unsur sebagaimana dimaksud pada ayat (3) yang dilakukan dalam bentuk pembajakan, dipidana dengan pidana penjara paling lama 10 (sepuluh) tahun dan atau pidana denda paling banyak Rp4.000.000.000,00 (empat miliar rupiah)

Daftar Isi

Daftar Isi.....	3
4. Learn ES6+.....	4
Learning Outcomes.....	4
Outline Materi.....	4
4.1. Introduction to ES6.....	5
4.2. Let & Const.....	6
4.2.1. Let.....	7
4.2.2. Const.....	11
4.3. Template Literal.....	12
4.4. Array Helper.....	13
4.4.1. Foreach.....	13
4.4.2. Map.....	14
4.4.3. Filter.....	15
4.4.4. Find.....	16
4.4.5. Every / Some.....	17
4.4.6. Reduce.....	18
4.5. Fat Arrow Function.....	19
4.6. Object Literal.....	20
4.7. Classes.....	22
4.8. Destructuring.....	25

4.

Learn ES6+

Learning Outcomes

Setelah selesai mempelajari bab ini, peserta mampu :

1. Mengetahui apa itu ES6.
2. Dapat menggunakan ES6 dalam membangun sebuah sistem pada JavaScript.

Outline Materi

1. Introduction to ES6
2. Let & Const
3. Template Literal
4. Array Helper
5. Fat Arrow Function
6. Object Literal
7. Classes
8. Destructuring

4.1. Introduction to ES6

ECMAScript adalah sebuah bahasa scripting yang di standardkan oleh Ecma International. ECMA sendiri adalah sebuah organisasi yang mengatur standarisasi sistem informasi dan komunikasi. Awalnya ECMA International memiliki nama European Computer Manufacturers Association yang pada tahun 1994 berganti ke nama yang digunakan saat ini untuk merepresentasikan bahwa organisasi ini sekarang bersifat global.



Spesifikasi ECMAScript adalah sebuah spesifikasi standard dari bahasa scripting yang dikembangkan oleh Brendan Eich. Awalnya bernama Mocha, lalu berganti nama menjadi LiveScript dan berganti nama lagi menjadi JavaScript seperti apa yang kita kenal saat ini. Pada Desember 1995 Sun Microsystem dan Netscape mengumumkan JavaScript.



Brendan Eich

Lima versi terakhir dari ECMAScript adalah:

Tanggal Release	Nama	Versi

Juni 2015	ECMAScript 2015 (ES2015)	6
Juni 2016	ECMAScript 2016 (ES2016)	7
Juni 2017	ECMAScript 2017 (ES2017)	8
Juni 2018	ECMAScript 2018 (ES2018)	9
Juni 2019	ECMAScript 2019 (ES2019)	10

ECMAScript 2015 atau ES2015 adalah versi 6 yang selanjutnya dikenal dengan ES6.

4.2. Let & Const

Sebelum ES6 dikenalkan, JavaScript hanya memiliki dua tipe scope: Global dan Function Scope. Maksudnya bagaimana global dan function scope? Global scope adalah variable yang dapat diakses darimana pun oleh program javascript sedangkan function scope adalah variable yang hanya bisa diakses pada function tertentu.

Berikut adalah perbedaan global dan function scope:

```
// Global scope
var carName = "Volvo";

// carName dapat digunakan disini

function myFunction() {
    // carName dapat digunakan disini
}

// carName dapat digunakan disini
```

```
// Function scope
```

```
// carName tidak dapat digunakan disini

function myFunction() {
  var carName = "Volvo";
  // carName dapat digunakan disini
}

// carName tidak dapat digunakan disini
```

4.2.1. Let

Pada ES6 dikenalkan cara pembuatan variable menggunakan let, dimana let sudah mendukung penggunaan secara block scope. Sebelumnya javascript hanya terdapat dua tipe scope yaitu global dan function.

Jadi untuk menggunakan block scope ini kita akan menggunakan let, jika kita menggunakan var maka kita tidak bisa menggunakan block scope. Apasih block scope itu?

Block scope adalah kondisi dimana variable yang dibuat didalam block tidak bisa diakses oleh script yang ada diluar block. Bagaimana script dapat dikatakan didalam atau diluar block? Nah block ini dibatasi oleh kurung kurawal { ... }. Bagaimana sih bentuk codenya?

```
// Contoh var

{
  var x = 2;
}

// variable x dapat digunakan disini.
```

```
// Contoh let

{
```

```
let x = 2;
}

// variable x tidak bisa digunakan disini.
```

Nah itu dia perbedaan antara var dan let dalam block scope. Dari masalah block scope diatas akan berimbas pada redeclaring variables atau saat pembuatan variable yang sama. Contohnya sebagai berikut:

```
// Contoh var

var x = 10;
// Disini nilai x adalah 10

{
  var x = 2;
  // Disini nilai x adalah 2
}

// Disini nilai x adalah 2
```

```
// Contoh var

let x = 10;
// Disini nilai x adalah 10

{
  let x = 2;
  // Disini nilai x adalah 2
}

// Disini nilai x adalah 10
```


Jadi saat menggunakan var, ketika di deklarasikan ulang pada block scope maka akan mempengaruhi global scope. Berbeda dengan let yang tidak berpengaruh pada global scope saat pendeklarasian ulang didalam block scope.

```
<!DOCTYPE html>
<html>
<body>

<h2>Declaring a Variable Using var</h2>

<p id="demo"></p>

<script>
var x = 10;
// Here x is 10
{
  var x = 2;
  // Here x is 2
}
// Here x is 2
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>

<h2>Declaring a Variable Using let</h2>
```

```
<p id="demo"></p>

<script>
let  x = 10;
// Here x is 10
{
  let x = 2;
  // Here x is 2
}
// Here x is 10
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Loop scope atau penggunaan scope pada saat melakukan looping. Contohnya:

```
var i = 5;
for (var i = 0; i < 10; i++) {
  // some statements
}
// Disini i adalah 10
```

Jika kita menggunakan var untuk melakukan looping, maka nilai i setelah looping block akan bernilai 10. Kenapa? Karena variable i di deklarasikan ulang saat looping dan mempengaruhi global scope.

Berbeda dengan penggunaan let, dimana global scope dan loop scope yang mirip seperti block scope tidak mempengaruhi satu sama lain:

```
let i = 5;
for (let i = 0; i < 10; i++) {
  // some statements
}
// Disini i adalah 5
```

4.2.2. Const

Variable yang dideklarasikan dengan menggunakan const memiliki sifat yang sama dengan let, namun const tidak bisa di deklarasikan ulang.

```
const PI = 3.141592653589793;  
PI = 3.14;           // Ini akan memunculkan error  
PI = PI + 10;        // Ini akan memunculkan errorr
```

Sama halnya dengan let, const juga dapat mendeklarasikan variable yang sama namun tidak mengganggu satu sama lain saat menggunakan block scope.

```
<!DOCTYPE html>  
<html>  
<body>  
  
<h2>Declaring a Variable Using const</h2>  
<p id="demo"></p>  
  
<script>  
const x = 10;  
// Disni niali x adalah 10  
{  
  const x = 2;  
  //Disini nilai x adalah 2  
}  
// Disni niali x adalah 10  
document.getElementById("demo").innerHTML = x;  
</script>  
  
</body>  
</html>
```

Saat menggunakan const, nilai harus di berikan ketika deklarasi variable. Jika tidak maka akan mengembalikan error.

```
// Contoh yang salah
const PI;
PI = 3.14159265359;
```

```
// Contoh yang benar
const PI = 3.14159265359;
```

4.3. Template Literal

Template literal adalah sebuah template string dimana kita dapat menanamkan sebuah expression. Tanpa template literal jika kita akan menuliskan sebuah string yang terdapat nilai variable nya akan terlihat seperti berikut:

```
let name = "Lukman Hakim";
let greeting = "Hello, nama saya adalah " + name + " saya berasal
dari kota Bandung.";
```

Namun dengan menggunakan template string kita dapat menulis sesuatu yang sama seperti contoh diatas tanpa memotong-motong scriptnya. Dengan bantuan syntax `${...}` untuk memasukan variable nya dan penggunaan back tick (``...``).

```
let name = "Lukman Hakim";
let greeting = `Hello, nama saya adalah ${name} saya berasal dari
kota Bandung.`;
```

Dari kedua contoh diatas akan mengembalikan hasil yang sama:

```
Hello, nama saya adalah Lukman Hakim saya berasal dari kota
Bandung.
```

Selain dapat langsung memunculkan string, template literal juga dapat mengolah ekspresi. Contohnya kita akan menampilkan sebuah hasil dari operator penjumlahan dengan menggunakan template literal.

```
let x = 10;
let y = 5;
```

```
let hasil = `Hasil dari penjumlahan ${x} dan ${y} adalah ${x + y}`;
```

Jangan lupa untuk menggunakan back tick (``) . Dari contoh diatas mengembalikan hasil sebagai berikut:

```
Hasil dari penjumlahan 10 dan 5 adalah 15
```

Selain penjumlahan kalian bisa melakukan operasi aritmatika lainnya seperti pengurangan, pembagian, perkalian dan sebagainya.

Template literal dapat memanggil sebuah function, dengan aturan yang sama mari kita lihat seperti apa:

```
function myFunction() {  
  return "Hello World";  
}  
  
console.log(`Pesannya adalah ${myFunction()}`);
```

Sekarang coba kalian cek console.log di browser masing-masing dan akan mendapati hasil sebagai berikut:

```
Pesannya adalah Hello World
```

4.4. Array Helper

Kita akan menggunakan array helper untuk memanipulasi data yang kita miliki. Dengan mengerti bagaimana karakteristik dari setiap helper maka kita akan lebih mudah dalam mengolah collection data.

Awalnya kalian akan menganggap setiap method mirip satu sama lain, tapi nanti kalian akan tahu mengapa menggunakan spesifik pada satu method dan bukan method lainnya.

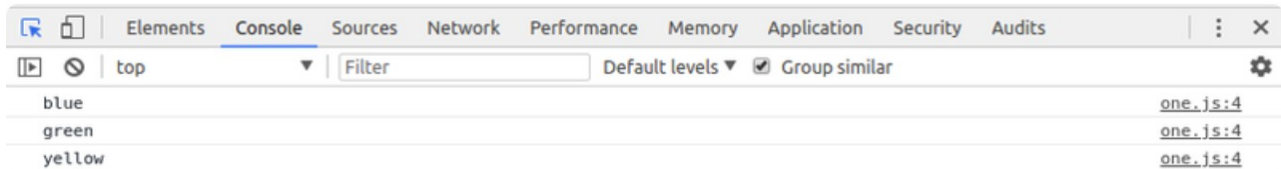
4.4.1. Foreach

Pertama-tama kita akan menulis simple for loop dan menampilkan semua value pada array.

```
let warna = ["blue", "green", "yellow"];
```

```
for (let i = 0; i < warna.length; i++) {
  console.log(warna[i]);
}
```

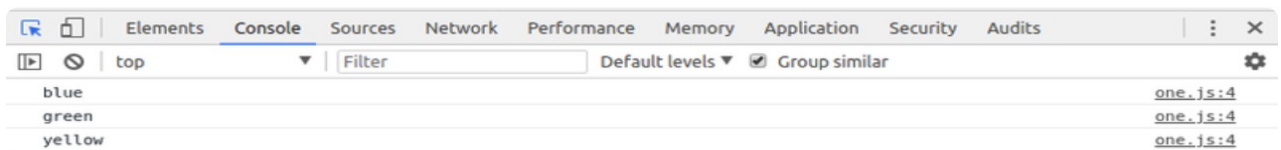
Maka akan menghasilkan:



Sekarang kita gunakan foreach:

```
let warna = ["blue", "green", "yellow"];
warna.forEach(function(color) {
  console.log(color);
});
```

Dari penggunaan foreach, hasilnya akan sama seperti for looping.



Foreach method yang dipanggil pada array akan mengambil setiap parameternya yang dikerjakan oleh fungsi iterasi. Fungsi akan berjalan sebanyak jumlah element pada array dan mengeksekusi kode yang berada didalam block function.

4.4.2. Map

Karena fungsinya sama dengan foreach, mari kita langsung saja mengapa map ini berbeda dengan foreach.

```
let numbers = [1, 2, 3, 4];

let n = numbers.forEach(function(number) {
  return number * 2;
});
```

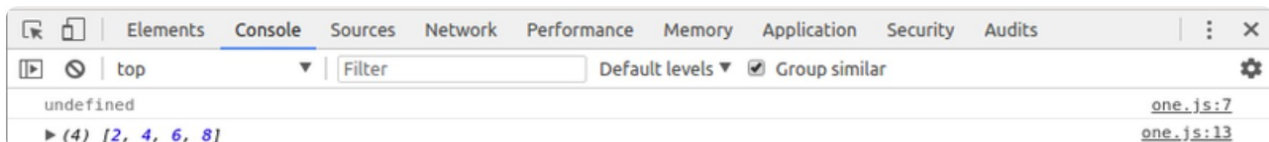


```
console.log(n);

let doubled = numbers.map(function(number) {
  return number * 2;
});

console.log(doubled);
```

Hasilnya akan seperti berikut:



Disini bisa kita lihat dengan menggunakan map kita membuat sebuah array baru, kita lihat lagi contoh diatas saat variable n yang menampung hasil dari foreach dipanggil, variable n memiliki value undefined sedangkan variable doubled yang digunakan untuk menampung hasil dari map akan memiliki value array baru.

Map digunakan ketika kita ingin nilai dari array yang sebelumnya tetap tidak ada perubahan. Contohnya ketika kita memiliki array numbers lalu yang pertama kita ingin mengalikan 3, lalu dengan array numbers yang sama (tanpa perubahan) kita ingin mengalikan 5 atau membaginya dengan 2, dengan menggunakan map array originalnya tidak akan pernah berubah karena disetiap proses map kita menghasilkan array baru.

4.4.3. Filter

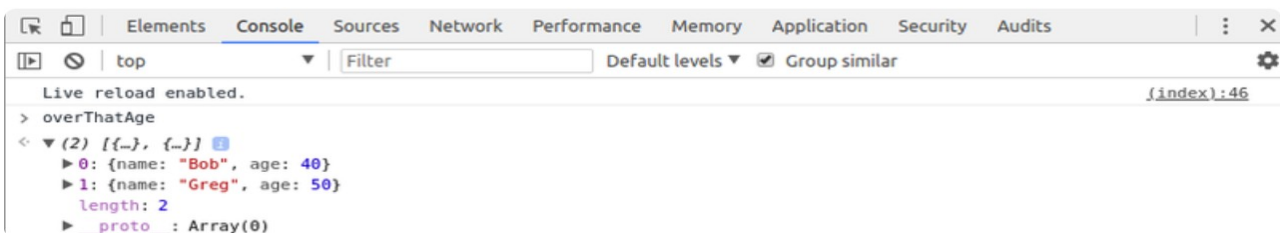
Filter adalah method yang melakukan pengulangan terhadap seluruh array. Namun filter menggunakan case yang didefinisikan secara strict atau ketat. Kita asumsikan jika kita memiliki sebuah array yang berisi object. Yang perlu diingat adalah dengan menggunakan filter method kita perlu mengembalikan statement yang memenuhi syarat dari kondisi yang kita tuliskan.

```
let people = [
```

```
{name: "Mark", age:10},
{name: "Tom", age:20},
{name: "Joe", age:30},
{name: "Bob", age:40},
{name: "Greg", age:50}
];

let overThatAge = people.filter(function(person) {
  return person.age > 35 && person.name === "Bob";
});
```

hasilnya akan seperti berikut :



Kita dapat melihat data yang sesuai dengan kondisi yang dituliskan, kita tidak lagi perlu untuk menggunakan if statement ketika menggunakan filter method. Kita juga tidak harus membatasi diri dengan satu kondisi. Walaupun pada kondisi yang kita tulis menggunakan logika AND namun pada filter ketika salah satu kondisi terpenuhi, maka data tersebut akan ter-filter.

4.4.4. Find

Fungsi utama dari penggunaan Find method adalah untuk melihat sebuah array dan mengambil nilai yang sesuai dengan kondisinya. Saat melakukan sebuah iterasi find akan melakukan pemeriksaan terhadap data yang masuk satu persatu, ketika data yang sesuai ditemukan maka iterasi akan dihentikan dan method akan mengembalikan hasil yang sesuai kondisi saja.

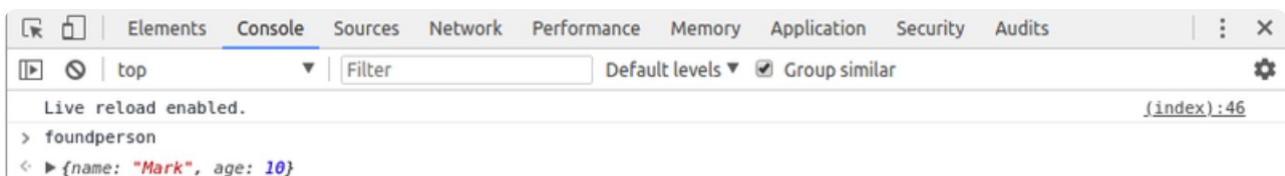


Penggunaan Find sangat membantu ketika kita menghadapi sebuah situasi dimana kita memiliki database yang besar, iterasi tidak harus berjalan seluruhnya ketika data yang kita cari sudah ditemukan membuat proses lebih cepat.

```
let people = [
  {name: "Mark", age:10},
  {name: "Tom", age:20},
  {name: "Joe", age:30},
  {name: "Bob", age:40},
  {name: "Mark", age:50}
];

let foundperson = people.find(function(person) {
  return person.name === "Mark";
});
```

Oke dengan data yang sama, mari kita ganti “Greg” menjadi “Mark” dengan seperti ini kita memiliki dua “Mark”. Selanjutnya kita gunakan find untuk mencari “Mark”, maka hasil dari Find Method adalah:



Find akan menghentikan iterasi ketika menemukan data yang sesuai dengan kondisinya, maka dari itu Mark dengan umur 10 lah yang dikembalikan oleh Find Method.

4.4.5. Every / Some

Every / Some method akan melakukan pemeriksaan terhadap seluruh value array dan mencocokkannya dengan kondisi yang diberikan. Kembalian dari method ini adalah sebuah Boolean yang menyatakan true atau false. Dimana true akan muncul jika seluruh value pada array cocok dengan kondisi sedangkan false akan muncul jika ada yang tidak cocok dengan kondisi yang diberikan.

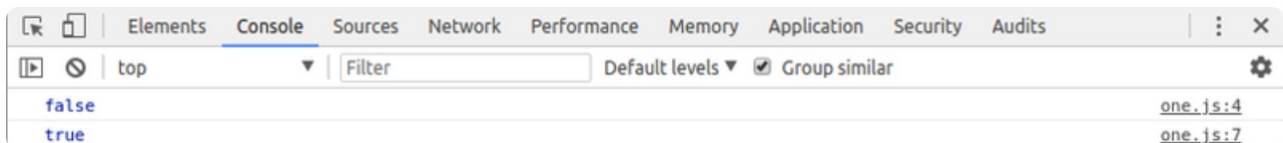


```
const names = ["Tom", "Joe", "John"];

let isEvery = names.every(function(name) { return name.length > 4 });
console.log(isEvery);

let areSome = names.some(function(name) { return name.length = 4 });
console.log(areSome);
```

Berikut adalah contoh Every dan Some, dimana setiap value array akan diperiksa apakah panjang karakter dari setiap value sama dengan 4.



Hasil dari Every mengembalikan nilai false sedangkan Some mengembalikan nilai true. Mengapa?

Every akan bernilai true pada kasus ini jika semua value array memiliki panjang karakter 4. Dengan kata lain nilai true pada every akan muncul jika semua value pada array sesuai dengan kondisi, jika ada yang tidak sesuai walaupun hanya satu maka hasilnya akan tetap false.

Some akan bernilai true pada kasus ini jika salah satu dari value array memiliki panjang karakter 4. Dengan kata lain nilai true pada some akan muncul jika salah satu dari setiap value array sesuai dengan kondisi, jika tidak ada yang cocok dengan kondisi maka hasilnya akan false.

4.4.6. Reduce

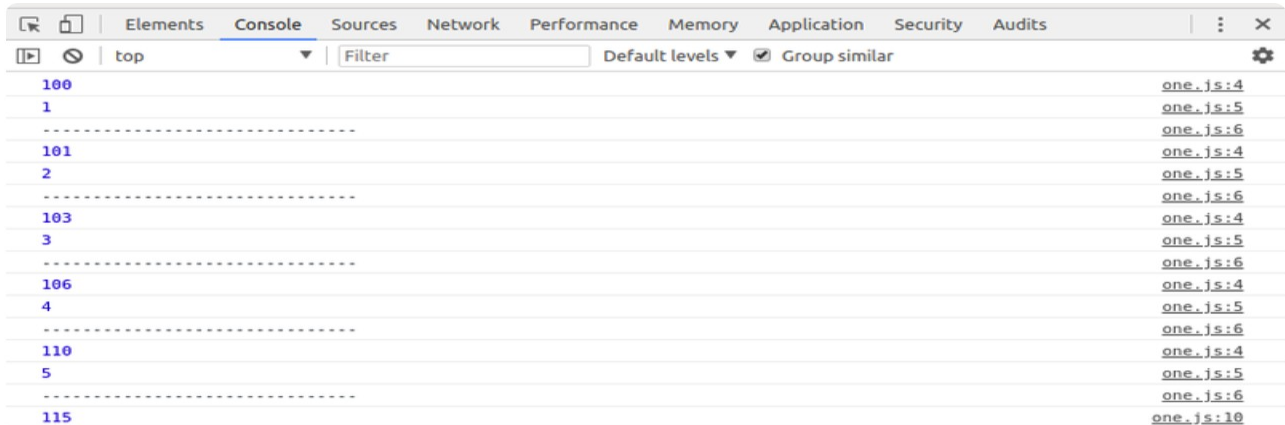
Untuk lebih memahami reduce, kita langsung saja ke contoh penggunaan dari method reduce ini:

```
myArray = [1,2,3,4,5];
```

```
let result = myArray.reduce(function(acc, val) {
  console.log(acc);
  console.log(val);
  console.log("-----");
  return acc + val;
}, 100);

console.log(result);
```

hasilnya sebagai berikut :



Log Message	Source
100	one.js:4
1	one.js:5
-----	one.js:6
101	one.js:4
2	one.js:5
-----	one.js:6
103	one.js:4
3	one.js:5
-----	one.js:6
106	one.js:4
4	one.js:5
-----	one.js:6
110	one.js:4
5	one.js:5
-----	one.js:6
115	one.js:10

Oke pada contoh diatas kita melakukan penambahan terhadap semua value dari array. Method reduce menggunakan dua variable

4.5. Fat Arrow Function

Kita sudah pernah menulis beberapa function selama pembelajaran ini, oke kita tulis sebuah function sederhana untuk menambahkan dua buah bilangan:

```
var add = function(a,b) {
  return a + b;
}
```

Benar, seperti itu bentuk function dan tidak ada yang salah. Lalu apa itu fat arrow function?

Oke pada ES6 memiliki sebuah fasilitas yang dapat membuat pekerjaan lebih mudah. Sekarang kita tulis function yang sama dengan bentuk fat arrow function:

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

Bisa kita lihat, terdapat dua perubahan pada penulisan fat arrow function. Yang pertama kita tidak lagi menuliskan keyword function dan yang kedua adalah kita menambahkan arrow function (=>) setelah parameter.

Bahkan kita dapat menuliskan fat arrow dalam bentuk yang lebih sederhana lagi, seperti berikut:

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

4.6. Object Literal

Membuat sebuah object pada beberapa bahasa pemrograman akan memakan waktu serta processing power karena class harus dideklarasikan sebelum semuanya dapat terpenuhi. Pada JavaScript sangat lah mudah dalam membuat sebuah object. Jika kalian masih ingat bagaimana cara membuat object maka bentuknya akan seperti berikut:

```
var myObject = {  
  prop1: 'hello',  
  prop2: 'world'.  
  output: function () {  
    console.log(this.prop1 + ' ' + this.prop2);  
  }  
};  
  
myObject.output(); // 'hello world'
```

Biasanya property object akan dibuat pada sebuah variable dengan nama yang sama, contohnya:

```
var
  a = 1, b = 2, c = 3;
  obj = {
    a: a,
    b: b,
    c: c
  };

// obj.a = 1, obj.b = 2, obj.c = 3
```

Dengan menggunakan object literal pada ES6 maka pembuatan object akan lebih mudah dimana kita tidak perlu menulis ulang property nya, yang perlu diperhatikan adalah penamaannya yang sama:

```
var
  a = 1, b = 2, c = 3;
  obj = {
    a
    b
    c
  };

// obj.a = 1, obj.b = 2, obj.c = 3
```

Kita juga dapat membuat object method dengan lebih mudah, berikut contoh normal jika kita membuat object method:

```
var lib = {
  sum: function(a, b) {return a + b;},
  mult: function(a, b) {return a * b;}
};

console.log(lib.sum(2, 3)); // 5
console.log(lib.mult(2, 3)); // 6
```

Pada ES6 kita bisa menghilangkan keyword function, jadi penulisan akan menjadi lebih singkat:

```
var lib = {  
  sum(a, b) {return a + b;},  
  mult(a, b) {return a * b;}  
};  
  
console.log(lib.sum(2, 3)); // 5  
console.log(lib.mult(2, 3)); // 6
```

Kita tidak bisa menggunakan fat arrow (=>) disini, karena jika kita menggunakan fat arrow maka kita perlu memberikan nama pada setiap functionnya. Jadi jika kita ingin menggunakan fat arrow bentuk syntax nya akan seperti berikut:

```
var lib = {  
  sum: (a, b) => a + b,  
  mult: (a, b) => a * b  
};  
  
console.log(lib.sum(2, 3)); // 5  
console.log(lib.mult(2, 3)); // 6
```

4.7. Classes

Sebelum kita membahas class pada javascript, satu hal yang perlu dijelaskan bahwa kelas pada javascript bukan lah fitur baru namun lebih ke pemanis seperti fat arrow, class ini fasilitas yang perlu diingat untuk mempermudah pekerjaan.

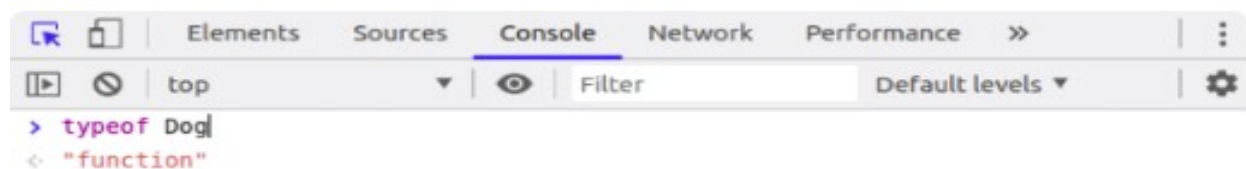
Sebenarnya javascript tidak benar-benar memiliki class inheritance, tetapi memiliki prototypal inheritance. Class inheritance telah diperkenalkan pada javascript karena faktanya pembuatan class terjadi pada sebagian besar bahasa pemrograman. Jadi apa yang dimaksud dengan class yang ada pada javascript, apakah tujuannya agar lebih mudah untuk membaca code yang ditulis? Jawabannya sederhana dan sama seperti bahasa pemrograman

lainnya. Class membuat object. Class seperti cetak biru atau dasar bagaimana object akan dibuat. Selain itu class pada javascript pada dasarnya adalah fungsi khusus.

Deklarasi kelas terdiri dari dua bagian, expression class, dan declaration class, seperti contoh berikut:

```
class Dog {  
  
}
```

Tidak ada yang sulit, tapi bagaimana jika kita melakukan console.log terhadap dog?



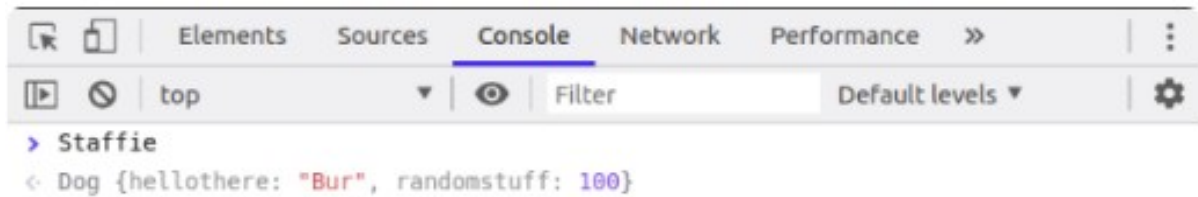
Seperti yang sudah dibahas sebelumnya, class merupakan function yang dijadikan dasar dalam pembuatan object.

Selanjutnya ada Constructor yang digunakan untuk menginisialisasi object data. Constructor dapat menerima beberapa parameter, pada kasus selanjutnya kita membuat dua parameter, name dan height, dan dengan menggunakan variable this kita akan memastikan kalau argument dapat menerima parameter dan menyimpannya pada variable.

Tentu saja kita bisa memberi nama yang berbeda untuk property, yang perlu diingat adalah nama parameter harus sama, namun itu tidak disarankan lebih baik nama property dan parameter memiliki nama yang sama:

```
class Dog {  
  constructor(name, height) {  
    this.hellothere = name;  
    this.randomstuff = height;  
  }  
}  
  
let Staffie = new Dog("bur", 100);
```

Berikut jika kita melakukan console.log pada Staffie:



Mari kita memberikan nama property yang sesuai, sehingga nanti menggunakan this.name untuk nama dan this.height untuk tinggi.

```
class Dog {
  constructor(name, height) {
    this.name = name;
    this.height = height;
  }
}

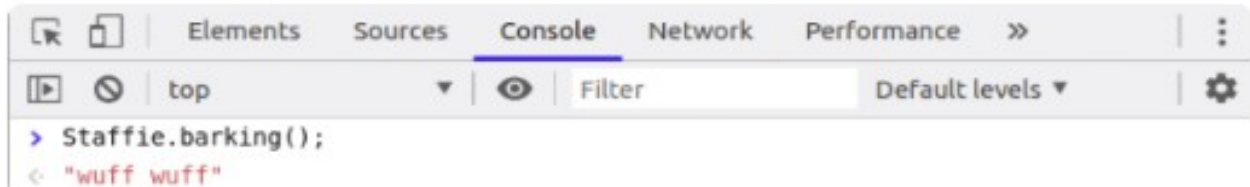
let Staffie = new Dog("bur", 100);
```

Kita dapat membuat sebuah method didalam class. Method ini sebenarnya adalah fungsi yang ditulis didalam class. Contohnya seperti berikut:

```
class Dog {
  constructor(name, height) {
    this.name = name;
    this.height = height;
  }
  barking() {
    return "wuff wuff";
  }
}

let Staffie = new Dog("bur", 100);
```


Sekarang variable Staffie memiliki method barking(), jika kita memanggil method barking() dengan console.log maka hasilnya akan seperti berikut:



4.8. Destructuring

Cara mudah untuk memahami destructuring adalah dengan contoh berikut:

```
var expense = {
  type: "Business",
  amount: "50$"
};
```

Kita memiliki object expense dengan property type dan amount. Bayangkan jika kita ingin mengambil value type dari expense kita akan melakukan hal berikut:

```
var expense = {
  type: "Business",
  amount: "50$"
};

var type = expense.type;
var amount = expense.amount;
```

Ini cara yang baik? Tentu saja, tidak ada yang salah namun kita dapat melakukannya lebih baik dengan menggunakan ES6. Caranya seperti berikut:

```
var expense = {
  type: "Business",
  amount: "50$"
};

const { type } = expense;
```

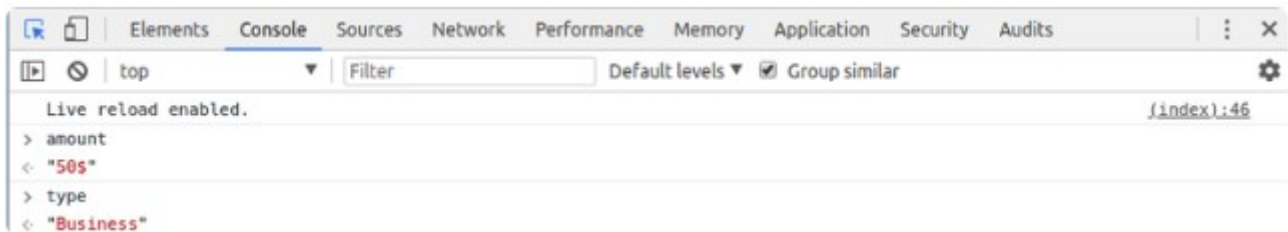
```
const { amount } = expense;
```

Dengan cara seperti ini kita akan mendapatkan hasil yang sama dengan cara penulisan normal. Code ini berarti dari object expense, kita akan meng-ekstrak type dan amount dan menyimpannya dalam sebuah variable yang dibuat dengan const dan memiliki nama yang sama.

Namun masih ada cara lain untuk menulisnya:

```
var expense = {  
  type: "Business",  
  amount: "50$"  
};  
  
const { type, amount } = expense;
```

Lalu coba kita lakukan console.log terhadap variable yang kita buat:



Nah itu dia destructuring dimana kita dapat mengambil value dari sebuah object.

