MUHAMMAD AZAMUDDIN

7 JURUS BARU Javascript

Table of Contents

7 Jurus Baru JavaScript

Maret, 2017

Halo, salam kenal, nama saya **Muhammad Azamuddin**, web developer, freelance author dan speaker. Saya bukan pakar, tapi karena kecintaan saya dengan dunia web development, alhamdulillah saya telah beberapa kali sharing terkait modern web development di beberapa beberapa kesempatan dianataranya di kantor Microsoft Indonesia dan Detik.com.

Kamu mungkin sekarang ini agak bingung melihat dunia javascript yang semakin aneh dengan sintak-sintak asing yang banyak digunakan. Nah sebenarnya semua itu tidak terlepas dari perkembangan dunia javascript ke arah yang lebih baik lagi. Untuk membantu kamu memasuki dunia javascript modern, saya mencoba untuk menulis ebook ini.

Beberapa tahun silam, Javascript dipandang sebagai bahasa mainan, sebagai hiasan saja untuk tampilan website, tidak lebih dari itu. Kini semua itu sudah berubah jauh. Javascript telah menjelma menjadi bahasa pemrograman yang sangat powerful untuk digunakan di dunia *development*. Baik untuk mengembangkan aplikasi web, mobile native, desktop, IoT maupun untuk keperluan lain. Dengan perkembangan yang sangat cepat itu, dunia javascript terus berubah menuju arah yang lebih baik. Mulai dari ketersediaan *library*, *framework*, module maupun tools lainnya untuk kemudahan developer, selain itu sintak dan fitur Javascript pun menjadi semakin baik.

Nah, jika kita menggunakan Javascript sekarang, umumnya *browser* yang kita pakai masih menggunakan standar EcmaScript 5th Edition, atau ES5 yang diterbitkan pada tahun 2009. EcmaScript merupakan sebuah standar yang memuat aturan-aturan atau spesifikasi dan fitur-fitur tertentu untuk sebuah bahasa pemrograman. Javascript termasuk bahasa yang mengimplementasikan standar ini. Kini, sudah tersedia versi yang lebih baru dari EcmaScript yang dikenal dengan EcmaScript 2015, EcmaScript 6th Edition atau orang biasa menyebut ES6.

ES6 merupakan kemajuan yang pesat bagi Javascript, beberapa fitur-fitur penting dan berguna kini tersedia untuk diimplementasikan, seperti Map, Promise, Arrow Function, dll. Namun, meskipun sudah disetujui sejak tahun 2016, belum semua *browser* mendukung fitur-fitur baru di ES6. Untuk itu beberapa developer mengembangkan *pre-compiler* agar kita bisa memanfaatkan fitur-fitur tersebut tanpa perlu khawatir script kita

tidak berjalan di beberapa browser. Cara kerja *pre-compiler* ini adalah dengan mengubah kode-kode yang kita tulis dengan fitur ES6 ke dalam kode ES5 pre-compile sehingga akhirnya tetap akan berjalan di semua browser yang mengimplementasikan EcmaScript versi sebelumnya, yakni ES5.

ES6 menawarkan beberapa fitur yang akan memudahkan developer javascript, siap belajar fitur-fitur tersebut? silahkan lanjutkan membaca.

Jurus 1: Arrow Function

Arrow function merupakan fungsi javascript yang bisa ditulis menggunakan sintak *fat arrow* (=>). Hal penting yang perlu dipahami adalah perbedaan referensi `this` antara sintak function biasa dengan sintak arrow function, yaitu jika pada ES5, `this` merujuk pada fungsi itu sendiri, di sintak arrow function keyword `this` merujuk pada *scope* di mana fungsi itu dipanggil.

```
var ruangan = {
    judul: 'Ruang Rapat 1',
    daftarPeserta: ['Joni', 'Taufik', 'Andri'],
    tampilkanPeserta(){

        /* sintak function biasa */
        this.daftarPeserta.forEach(function(peserta){
            console.log(peserta + ' ikut rapat di ' + this.judul);
            // Hasil console => Joni ikut rapat di undefined, dst
            // this.judul menjadi "undefined" karena "this" merujuk pada anynomous functio
n(peserta){}
        })

        /* sintak arrow function */
        this.daftarPeserta.forEach( (peserta) => {
            console.log(peserta + ' ikut rapat di ' + this.judul)
            // Hasil console => Joni ikut rapat di Ruang Rapat 1, dst
            // this.judul memiliki nilai "Ruang Rapat 1" karena "this" tidak merujuk pada
arrow function, tapi
            // merujuk pada scope dimana arrow function itu dipanggil, yaitu object "ruang
an".
        })
    }
}
```

Selain itu, arrow function bisa ditulis dengan *expression body* maupun statement body. Contoh:

```
// expression
daftarPeserta.filter(x => "Taufik")

// statement
daftarPeserta.filter(x => {
    return x == "Taufik"
})
```

Shared arguments. Sintak arrow function juga memiliki akses ke arguments fungsi yang memanggilnya. Lihat berikut ini

```
function square() {
  let example = () => {
    let numbers = [];
    // arguments merupakan arguments yang diassign ke fungsi square, dan arrow function ex
ample di sini
    // memiliki akses langsung ke arguments tersebut.
    for (let number of arguments) {
      numbers.push(number * number);
    }

    return numbers;
  };

  return example();
}

square(2, 4, 7.5, 8, 11.5, 21); // returns: [4, 16, 56.25, 64, 132.25, 441]
```

Jurus 2: Let & Constant

Let

let is the new var. let merupakan cara baru untuk mendefinisikan variable, bisa dikatakan sebagai alternatif dari var. Perbedaannya adalah pada *scope*, var bisa diakses oleh function scope terdekat, sementara let hanya bisa diakses oleh *block scope.*

Contoh Kode:

```
/**
 * GLOBAL SCOPE
 * Jika let ditulis di Global Scope. Tidak di dalam function block
 */

let me = 'go';  // globally scoped
var i = 'bisa'; // globally scoped

console.log(window.me); // undefined
console.log(window.i); // 'bisa'


/**
 * FUNCTION SCOPE
 * jika ditulis dalam function scope, keduanya sama persis
 */
function ingWithinEstablishedParameters() {
    let greeting = 'awesome worker!'; //function block scoped
    var cheer = 'go!'; //function block scoped
}



/**
 * BLOCK SCOPE
 * Ini dia perbedaannya.
 * let hanya bisa diakses oleh block in for() loop, sementara var bisa diakses dari scope
function
 */

function makan() {
    // makan tidak bisa diakses di sini

    for( let bakso = 0; bakso < 3; bakso++ ) {
```

```
        // bakso hanya bisa diakses oleh ( dan di dalam in for() loop)
        // dan selalu ada variable bakso sendiri-sendiri untuk setiap perulangan (loop)
    }

    // bakso tidak bisa diakses di sini
}

function minum() {
    // kopi tidak bisa diakses di sini

    for( var kopi = 0; kopi < 5; kopi++ ) {
        // kopi bisa diakses di sini dan di seluruh function block ke bawah
    }

    // kopi sekarang bisa diakses dari sini
}
```

Hal yang perlu diperhatikan tentang let

- dalam mode 'strict', variabel harus dideklarasikan terlebih dahulu.
  - `let x = 3; console.log(x)`
  - `console.log(y)` error karena y belum dideklarasikan
- jika menggunakan mode strict, tidak boleh mendeklarasi ulang variable yang sudah diinisiasi dengan let
  - `let x = 4; let x = 5` error
- let sama seperti var, bisa kita ubah nilainya setelah dideklarasi
  - `let x = 4; x = 5` boleh

Constant

jika let bisa kita ubah nilainya setelah dideklarasikan, const tidak bisa diubah begitu dideklarasikan

```
function f() {
  {
    let x;

    {

      // boleh, karena x di bawah ini berada dalam block berbeda dari let x di atas.
      const x = "sneaky";

      // error, x baru saja didefinisikan menggunakan const, dan masih dalam block yang sa
ma.
```

```
      // constant tidak boleh diubah nilainya setelah dideklarasikan
      x = "foo";
    }

    // boleh, mengubah nilai x yang dideklarasikan menggunakan let
    // x di bawah ini bukanlah x yand dideklarasikan dengan const
    x = "bar";

    // error,let tidak boleh dideklarasi ulang, lihat let x di bagian paling atas, masih s
atu block scope
    let x = "inner";
  }
}
```

Default, Rest, and Spread Properties

Default params

Kini dalam javascript kita bisa mendefinisikan *default* params untuk sebuah fungsi.

```
function f(x, y=12) {
  // y memiliki nilai default 12 jika tidak diassign pada saat pemanggilan fungsi f
  return x + y;
}

// contoh, default untuk y adalah 12
// karena y tidak diisi maka y = 12, 2 + 12 = 15
f(3) == 15 // true
```

Rest Properties

Rest kalo diartikan secara bebas artinya "sisa" atau "sisanya". Misal kita punya 3 params, trus kita define param pertama sebagai `x` lalu 2 artikel selanjutnya kita define sebagai `y` . Brrt param pertama = `x`, sisanya (*rest*) adalah `y`. Kita tulis dengan sintak ... (titik tiga).

contoh kode:

```
function f(x, ...y) {
  // y menjadi Array
  return x * y.length;
}
f(3, "hello", true) == 6 // true
// "hello" dan true ditangkap di function f sebagai ["hello", true], gampang kan?
```

Spread Properties

Spread kalo diartikan secara bebas artinya "sebar" atau "sebarkan". Misal kita punya arguments array [1,2,3] sebar menjadi (1,2,3). Sintaknya adalah dengan ... (titik tiga) juga.

contoh kode:

```
function f(x, y, z) {
  return x + y + z;
}

// Assign setiap item di Array menjadi arguments, x,y,z
// kebalikan rest properties
f(...[1,2,3]) == 6 // true
```

Rest -> dari daftar arguments menjadi array `f(3, "hello", true)` menjadi seolah-olah `f(3, ["hello", true])`

Spread -> dari array menjadi daftar arguments `f(...[1,2,3])` menjadi seolah-olah `f(1,2,3)`

Template String

Template string merupakan sintak pemanis untuk membuat string yang terstruktur. Mirip dengan fitur penyisipan variable ke string pada Perl, Python dan bahasa lainnya.

```
// Template string dasar
`Ini merupakan template string `


// Multiline string
` Di ES5 string seperti ini bisa menyebabkan
  error tapi di ES6 tidak`

// Interpolate variable bindings
// Menyisipkan variabel
var nama = "Azam", waktu = "hari ini"
`Halo ${nama}, apa kabarmu ${waktu}?`
//Hasil => Halo Azam, apa kabarmu hari ini?
```

Class

ES6 Class merupakan cara yang lebih praktis dalam membuat class dibandingkan prototype-based OO di ES5. Sintak class ini menjadikan class di javascript lebih mudah untuk dipakai dan di kelola. Class tetap memiliki dukungan terhadap prototype-based inheritance, dan kini mempunyai kemampuan untuk memanggil super(), memiliki instance, method statis (static method) dan constructor.

```
// Mendefinisikan class

class Hewan {
    constructor(nama){
        this.nama = nama;
    }

    suara(){
        console.log('Graaar')
    }

    kenalan(){
        console.log(`nama saya ${this.nama}`)
    }

    jumlahKaki(){
        console.log(4)
    }

    // static method
    static sapa(){
        console.log('Helloooo')
    }
}


// Inheritance
class Kucing extends Hewan {
    constructor(nama){
        // panggil constructor method dari parent class (class Hewan)
        super(nama)
    }

    // override method suara pada parent class (class Hewan)
    suara(){
        console.log('Meaaow')
    }
}
```

```
// buat instance untuk masing-masing class
var hewan = new Hewan('Helli')
    hewan.suara() // Graaar
    hewan.jumlahKaki() // 4
    hewan.kenalan() // nama saya Helli

var jumbo = new Kucing('Taucho')
    jumbo.suara() // Meaaow
    jumbo.jumlahKaki() // 4
    jumbo.kenalan() // nama saya Taucho


// pemanggilan method statis
    Hewan.sapa()
```

Modules

Definisikan module kita, misal berada di path lib/math.js. Kita akan membuat fungsi tambah dan variabel pi kemudian kita jadikan module itu bisa digunakan oleh file js lain. Gunakan export;

```
// PATH: lib/math.js
export function tambah(x, y) {
  return x + y;
}
export var pi = 3.141593;

export function kurang(x,y){
  return x - y
}
```

Lalu, file app.js ingin menggunakan module math.js yang baru saja kita buat di path lib/math.js

```
// PATH: app.js

// import semua fitur (fungsi tambah dan variabel pi) menggunakan * as math  sebagai math
import * as math from "lib/math";

// gunakan math dalam logic app kita
console.log("2π = " + math.tambah(math.pi, math.pi));
```

Contoh, kode lain ingin mengakses module math juga

```
// PATH: otherApp.js
// kali ini kita tidak ingin import semua fitur, hanya ambil fungsi tambah dan variable pi
 saja
import {tambah, pi} from "lib/math";

// gunakan fungsi sum dan pi yang baru saja kita import dari lib/math.js
console.log("2π = " + tambah(pi, pi));
```

Promise

Promise jika diartikan secara bebas menjadi "janji". Janji adalah sesuatu yang ingin kita lakukan nanti, meskipun kita harus menepati janji, terkadang ada halangan di luar kendali kita yang menjadikan kita gagal dalam menepati janji tersebut. Jika kita bisa menepatinya, maka dalam konteks ES6 ini promise kita telah resolved, sementara jika gagal menepati janji maka promise kita rejected.

```
function timeout(duration = 0) {
    return new Promise((resolve, reject) => {
        setTimeout(resolve, duration);
    })
}

var p = timeout(1000).then(() => {
    return timeout(2000);
}).then(() => {
    throw new Error("hmm");
}).catch(err => {
    return Promise.all([timeout(100), timeout(200)]);
})
```

Sudah paham apa masih perlu saya jelaskan dengan lebih simpel lagi? baik silahkan lihat contoh di bawah

```
// buat promise object
var janji = new Promise((resolve, reject)=>{
  setTimeout(reject, 4000)
})

// kita janji ke teman kita mau ngajak makan
let makan = janji.then(()=>{
  console.log('Makan ditepati')
}).catch(()=>{
  console.log('Makan ga jadi')
})

// coba ganti kode "setTimeout(reject, 4000)" menjadi "setTimeout(resolve, 4000)"
// Silahkan utak atik sendiri di sini http://jsbin.com/heroteyoli/edit?html,js,console
```

Penutup

Alhamdulillah, kamu baru saja mempelajari 7 jurus baru javascript. Semoga yang kamu dapat ini bisa membuatmu lebih siap untuk menjadi modern web app developer. Ingat apa yang baru kamu pelajari di buku ini merupakan pembuka mata, dasar yang kamu perlu tahu. Jangan sampai kamu berpuas diri, karena masih banyak ilmu yang bisa kamu dapatkan, dalami materi-materi di buku ini dan praktikan jurus-jurusnya, kalau sekarang belum bisa menerapkan jangan khawatir, jurus ini akan berguna ketika kamu benar-benar nyemplung ke dunia pengembangan web app dengan javascript baik di browser maupun dengan node.

- Like FB Page saya **https://facebook.com/arrowfunxtion**
- Anda juga bisa follow channel telegram **https://t.me/arrowfunxtion**

Salam, kawanmu

Muhammad Azamuddin ^_^

Acknolwedgements

Cover Karate : Design by Freepik - http://www.freepik.com/free-vector/polygonal-karate-expert_728434.htm