**JAVASCRIPT LANJUTAN | 9. Asynchronous Javascript**

Khusunya konsep Asynchronous pada Javascript, kita akan mulai dulu dari definisi, konsep, dan penjelasan nya. Nah tujuan kita belajar Asynchronous ini adalah agar kedepannya kita tidak bingung ketika dihadapkan dengan teknik-teknik programming Asynchronous pada javascript, seperti :

* Callback
* Promise
* Ajax
* Async & await

Jadi sebelum masuk membahas mengenai topik-topik di atas ada baik nya pahamin betul konsep Asynchronous.

Kita mulai balik lagi bertanya pada diri kita apasih javascript itu? Kalo kalian baru belajar javascript atau kalian pak dhika ajar javascript ke mahasiswa nya itu ketika kita baru belajar HTML dan CSS itu saya jelasin JavaScript tuh gini :

**JavaScript** *“Bahasa pemrograman tingkat tinggi dan dinamis yang secara umum dapat berjalan pada browser.”* . <https://id.wikipedia.org/wiki/JavaScript>

Nah itu penjelasan untuk orang awam yang belum tau tentang javascript, Nah tapi karena kita udah masuk ke javaScript lanjutan pemahaman tentang javscript itu bisa jadi lebih kompleks ya, Definisi javascript sekarang jadi gini :

**JavaScript** *“is* a **single-threaded, non-blocking***,* **asynchronous** *and* **concurrent** *language.”* <http://latentflip.com/>

**Single-threaded**

Jadi simpelnya **Thread** tuh gini *“Urutan eksekusi kode yang dapat dilakukan secara bebas / idependent satu sama lain.”* . Jadi intinya javascript hanay bisa melakukan satu hal pada satu waktu tertentu karena dia Single-threaded.

Jadi contohnya gini javascript Cuma punya satu thread ( thread 1) kalo dia punya satu Task -1, maka setelah Task nya selesai dia baru bisa menyelesaikan Task yg ke-2 ( Task – 2 ) setelah task 2 selesai baru bisa lanjut ke task berikut nya dan task berikut nya lagi, gini :

Thread 1 Task – 1 > Task - 2 > Task – 3 > dan seterus nya

Anggap aja kalo bingung javascript itu membaca baris perbaris ada tahapan nya dan menyelesaikan nya gitu aja.

**Multi-threaded**

Sedangkan kalo multi threaded nah thread nya bisa banyak walaupun secara umum tetep aja pekerjaan nya hanya bisa dilakukan satu-satu tapi bisa aja dikerjakan oleh thread yg berbeda, contohnya gini :

Thread 1 Task – 1 Task - 4.

Thread 2 Task - 2

Thread 3 Task – 3

Nah jadi pekerjaan nya bisa dibagi-bagi kedalam Thread yg bebeda itu kalo multi threaded, nah sedangkan yg kedua nih Blocking dan NonBlocking.

**Blocking**

Jadi kalo Blocking gini ketika kita punya satu buah Task atau script gitu ya pada saat di eksekusi ternyata script ini membutuhkan waktu yang lama kalo Task 1 belum selesai maka dia gabisa nyelesaiin Task-2 nya dong kan harus nunggu task-1 nya selesai dulu baru dia bisa lanjut ke Task berikut nya.

**Non-blocking**

Nah kalo Non-Blocking itu bisa gini ketika kita mengerjakan satu hal walaupun pekerjaan nya belum beres kita bisa lanjut ke Task berikut nya, jadi misal Task-1 belum selesai tapi kita mau lanjut ke Task-2 itu bisa walaupun Task-1 belum selesai.

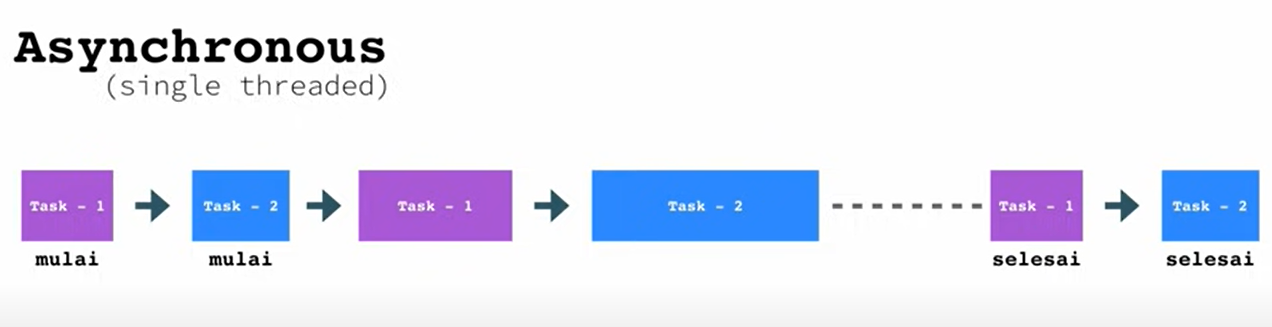
**Synchronous**

Nah sedangkan Asynchronous dan Synchronous itu begini, kita mulai dari Synchronous, jadi Synchronous itu yang biasa, gini :

Task – 1 > Task - 2 > Task – 3 > Task – 4

**Asynchronous (***single threaded***)**

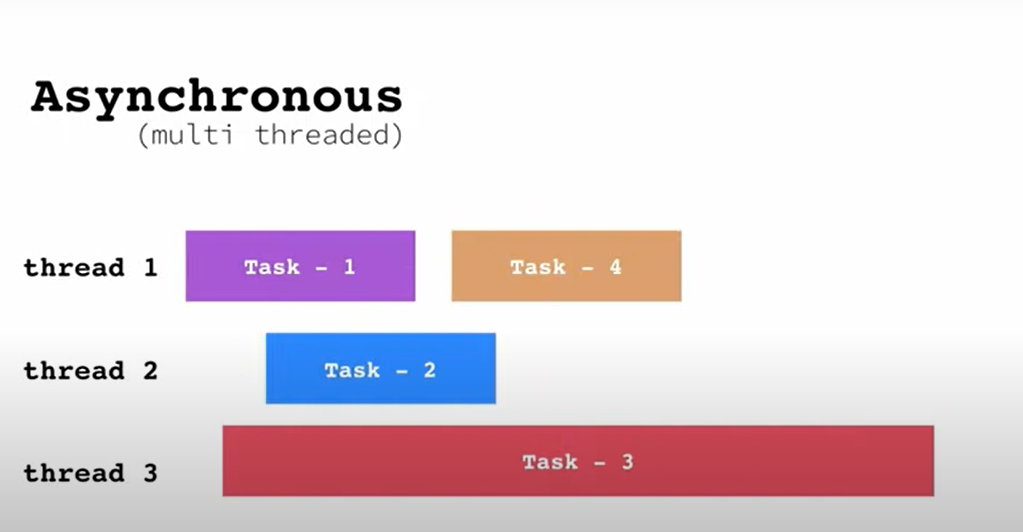
Sedangkan Asynchronous gini misalnya kita mengerjakan Task-1 yg pertama itu mulai mengerjakan, nah Taks-1 belum beres tapi Thread nya bisa pindah ke Task yg lain misal Task-2 mulai nya, padahal Task pertama belum selesai jadi dia pindah dulu terus nanti Task-2 belum selesai pindah lagi ngerjain Task-1 ngerjain Task-2, sampai akhirnya task yg ke-1 selesai dan Task yg ke-2 selesai, gini gambaran nya :



Nah itu kalo misalkan Asynchronous nya Single-Threaded ya jadi pindah-pindah gitu.

**Asynchronous (***multi threaded***)**

Sedangkan kalo Asynchronous multi threaded ini lebih keren lagi ada 3 thread misalnya kita bisa ngerjainnya bareng-bareng gitu, gini gambarang nya :



Nah jadi konsep itu kalo kita gabungkan misalkan kita punya (Asynchronous + Single-threaded = Concurrency) Asynchronous yg single-threaded itu kita sebutnya sesuatu yg Concurrency, sedangkan kalo (Asynchronous + Multi-threaded = Parallelism) Asynchronous dan multi-threaded itu kita sebut nya Parallelism jadi benar-benar dikerjakan secara bersamaan dalam satu waktu terntentu, jadi itu adalah isitlah-istilah yg ada pada penjelasan Javascript tadi.

**Kesimpulan**

* **Single vs Multi Thread**

Lingkungan eksekusi ‘task’

Jadi kesimpulan nya kalo kita ngomongin Single dan Multi Thread itu ada hubungan nya dengan Lingkungan eksekusi ‘task’ nya yg dikerjain nya menggunakan 1 thread atau banyak thread.

* **Blocking vs Non-Blocking**

Teknik ‘ngoding’ (10 related)

Sedangkan Blocking dan Non-Blocking itu adalah teknik ngoding kita yg biasanya ada hubungan dengan input dan output, jadi ketika tombol diklik, ketika kita memilih sesuatu, ketika kita mengerjakan sebuah fungsi yg kompleks jadi itu teknik ngoding.

* **Synchronous vs Asynchronous**

Teknik ‘ngoding’ (HTTP Request related)

Sebenernya mirip banget dengan Synchronous dan Asynchronous tapi untuk Synchronous dan Asynchronous itu, itu adalah teknik ngoding yg ada hubungan nya dengan request kita terhadap HTTP, jadi biasanya itu kalo kita mau menggunakan AJAX.

* **Synchronous vs Asynchronous**

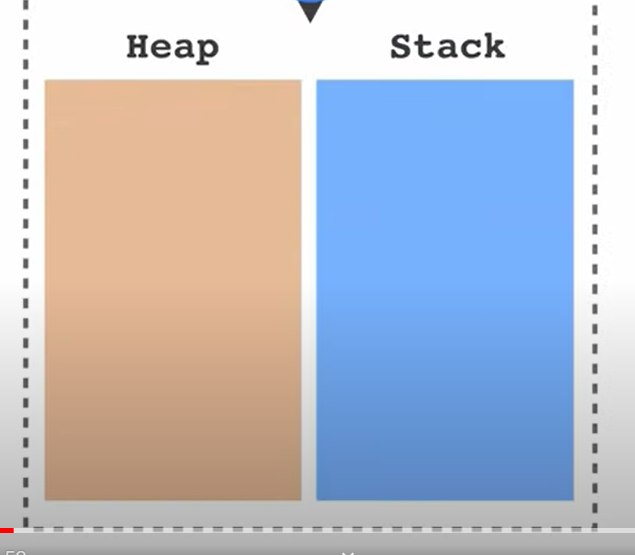
Lingkungan eksekusi ‘task’

Dan sedangkan Concurrent dan Parallel juga sama itu hubungan nya dengan lingkungan eksekusi ‘task’ juga biasa nya kalo Concurrent dan Parallel ada hubungan nya sama insfrastruktur ada hubungan nya sama hadwer atau virtualisasi, tapi kita fokusnya di **Non-blocking** dan **Asynchronous** aja dulu.

Kalo kalian agak bingung coba kita liat lebih dalam pokoknya kalian tahu dulu sekarang javascript itu Single-threaded jadi sebenernya dia Cuma bisa melakukan satu hal dalam satu waktu tertentu tapi dia juga bisa **Non-blocking** dan **Asynchronous**.

Sekarang kita liat didalam JavaScript itu ada sesuatu yg namanya “V8 Engine” atau mesin yg jalanin Javascript itu namanya “V8 Engine”, jadi bukan Cuma mobil yg punya mesin V8, jadi ada mesin V8 yg ditanam kedalam web browser yg kita liat isi nya didalam nya tuh kek gini.

Jadi V8 itu terdiri dari 2bagian ada yg namnya Heap ada yg namanya Stack, gini :



Sebetulnya kita ini sebagai developer apalagi yang baru belajar gaperlu tau detail tentang itu ya, jadi itu hanya wawasan aja supaya nanti kalian lebih paham mengenai Asynchronous ya, nah Head itu digunakan untuk Allocation Memory ya digunakan untuk tempat penyimpanan variable terus juga function yg hubungan nya dengan hoisting, kalo kalian masih ingt kita pernah belajar hoisting itu dilakukan didalam bagian Heap.

Nah sedangkan Stack itu Javascript gunakan untuk pemanggilan (function Call & Execution Context) function dan eksekusi script nya, jadi kalo variable dan function disimpan nya didalam Heap tapi ketika dipanggil dan variable nya dieksekusi misalnya dicetak ke layar itu dilakukan nya di Stack.

Kita akan fokus dulu di Stack ya karena di stack itu yg ada hubungan nya dengan Asynchronous yg kita pelajari, jadi contoh nya kalo kita punya script yg simpel aja gini :

        console.log('satu');

        console.log('dua');

        console.log('tiga');

itu adalah eksekusi berarti ada nya di Stack bukan di Heap, nah yg terjadi gini pada saat javascript nya membaca baris pertama dia melihat ada console.log(‘satu’) maka console.log itu akan di eksekusi didalam Stack, lalu kalo sudah selesai di eksekusi perintah console.lo hilang didalam stack nya, lalu javascript nya melanjutkan kebaris berikut nya ketemu console.log lagi masuk kedalam stack setelah selesai dieksekusi hilang dia dari Stack dan begitu seterusnya.

Jadi dia akan masuk ke Stack saat di eksekusi setelah selesai eksekusi maka kode program console.log itu misalnya akan hilang dari Stack karena sudah selesai di eksekusi kode console.log nya. Itu kalo script nya sesimpel console.log ya.

Nah sekarang kalo secript nya seperti ini :

        function kalikan(*a*, *b*) {

            return *a* \* *b*;

        }

        function hitungLuas(*sisi*) {

            return kalikan(*sisi*, *sisi*);

        }

        function cetakLuasKotak(*sisi*) {

            const luas = hitungLuas(*sisi*);

            console.log(luas);

        }

        cetakLuasKotak(5);

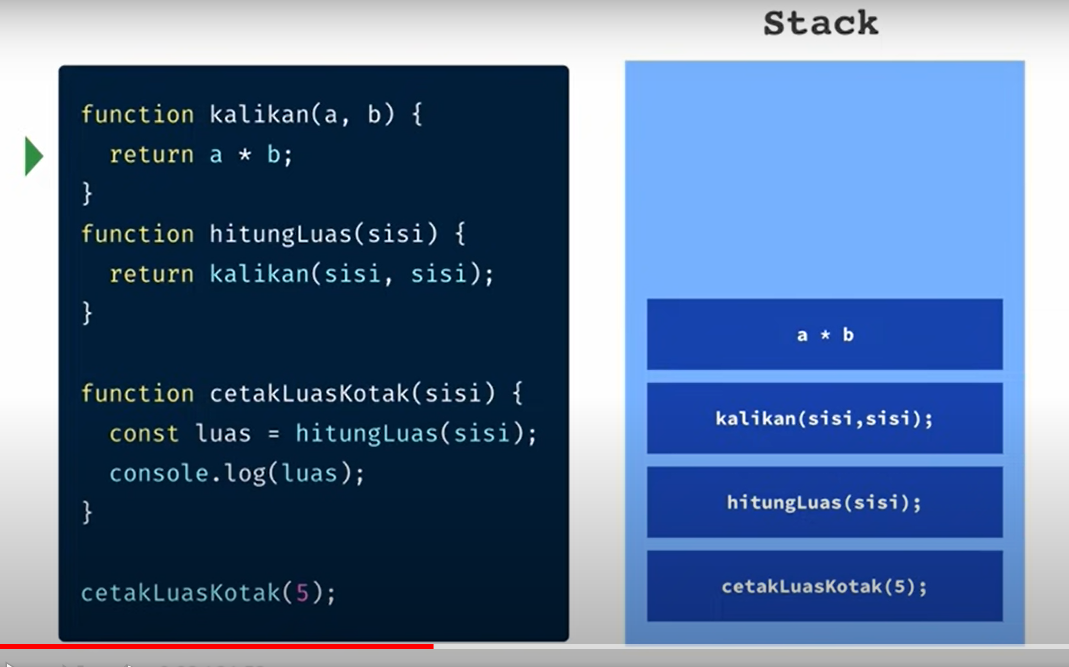
Jadi ada function kalika() yg menerima 2 argument a dan b, lalu mengalikan a dan b lalu ada function hitungLuas yg mengembalikan function kalikan(), ada lagi function cetakLuasKotak() yg isi nya membuat variable namanya luas yg isi nya memanggil function hitungLuas jadi saling memanggil ya.

Nah coba dimana Javascript menjalankan code kita ini, mulai dari atas dulu ya kita nulis nya kaliakan() dulu itu masuk kedalam Heap di hoisting ya, begitu pula hitunglLuas() dan cetakLuasKotak() juga masuk kedalam Heap di hoisting.

Nah lalu masuk ke memanggil fungsi cetakLuasKotak(5) dia mengeksekusi, nah kalo eksekusi dia masuk kedalam Stack, berarti sekarang dia nyari ada ga function kalo ada masuk kedalam nya dia bikin variable terus variable nya disimpan di HEAP terus jalanin fungsi hitungLuas() nah itu eksekusi lagi ya berarti dia masuk ke STACK jadi mulai numpuk nih karena yg memanggil cetakKotakLuas(5) itu dia belum selesai jadi nya belum hilang.

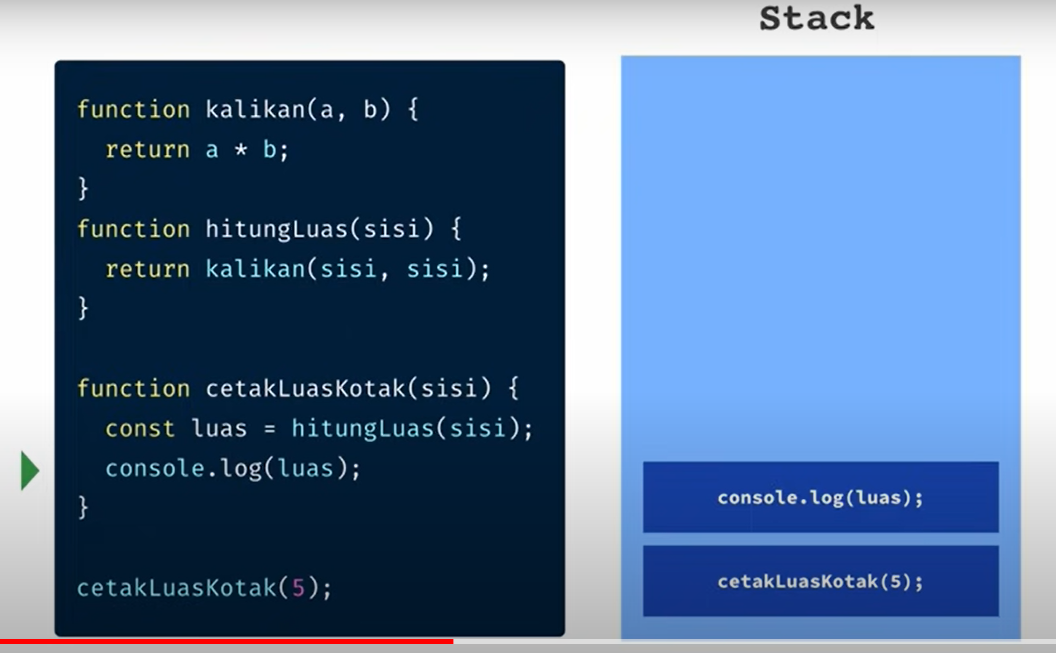
Nah setelah itu ga turun ke consol.log dulu ya karena hitungLuas() harus kita eksekusi dulu berarti cari function hitungLuas(), lalu fungsi itu mereturn fungsi kalikan() lalu masuk ke eksekusi berarti tambah lagi STACK nya jadi ada 3 taks lah gitu.

Nah masuk ke fungsi kalikan() ekekusi return a dan b jadi kita eksekusi didalam STACK nya (a \* b), a kali b itu tadi 5 ya berarti 5 kali 5 = 25 ya, gini :



Nah setelah sudah dikalikan perlahan satu persatu hilang task nya, pertama (a \* b) nya hilang di STACK dari tumpukan nya, lalu kembali ke return kalikan(sisi, sisi) hasil nya 25 ya lalu hilang dari tumpukan di STACK nya, terus turun lagi ke hitungLuas() hasilnya 25 ya berarti variable luas isi nya 25 selesai hitungLuas() hilang dari tumpukan STACK.

Nah sekarang dia turun ke console.log(luas) lalu console.log nya masuk ke eksekusi berarti ada tumpukan lagi di Stack nya, gini :



Nah tampil console.log(luas) yg hasilnya 25 lalu udah selsai dia hilang dari STACK dan cetakLuasKotak() juga udah selesai maka hilang dari stack nya, jadi gitu ya cara baca nya menelusurinya pelan-pelan aja.

Lebih detail kunjungi : <http://latentflip.com/loupe/> by. Phillip Roberts

Nah itu berdasarkan presentasi dia yg keren banget, dia certia mengenai apa itu event loop didalam Javascript youtobe : <https://www.youtube.com/watch?v=8aGhZQkoFbQ>

Kembali ke Stack

Nah kalian udah tau kalo kasus yg pertama yg console.log(‘satu’) gitu aman ya, nah gimana kalo sekarang kasus nya gini :

        console.log('satu');

        prosesYangSuperKompleks();

        console.log('tiga');

Jadi kita ada console.log(‘satu’) lalu dibawah nya kita menjalankan sebuah function yg cerita nya function itu menjalankan sesuatu yg kompleks banget misalnya request data ke API atau misalnya img prosessing yg lama gitu ya, baru selanjutnya tampilin consoe.log lagi.

Nah ini harus nya kalian udah bisa nebak jadi kalo console.log(‘satu’) itu ke eksekusi berarti didalam stack ya, lalu tampil dan hilang console.log dari tumpukan Stack nya cepet gitu aja, tapi begitu masuk ke function yg kompleks misalnya nah dia harus nunggu dulu sampe proses nya selesai dan function yg kompleks nya hilang dari stack baru console.log(‘tiga’) tampil.

Jadi semakin lama proses kompleks itu console.log(‘tiga’) nya semakin lama tampilnya nah itu salah satu contoh dari Blocking, blocking ini memungkinkan ditampilan web nya kita gabisa melakukan apa-apa, kita gabisa klik tombol kita gabisa ngisi input karena proses nya masih berjalan, mungkin ini kalian sebutnya ngeleg atau loading gitu ya.

Nah itu padahal proses berjalan tampilan nya keBlok nah ini tentu saja tidak kita ingin kan jadi kita harus nunggu dulu sampai proses nya selesai baru deh mengeksekusi code selanjutnya.

Nah cara menangani nya kita bisa lakukan sesuatu yg disebut dengan Asynchronous Callback apa itu nanti dijelaskan nya divideo yg lain, tapi intinya kita bisa akalin dengan melakukan sesuatu seperti ini :

        console.log('satu');

        setTimeout(() => {

            prosesYangSuperKompleks();

        }, 0);

        console.log('tiga');

Jadi kita panggil sebuah timing function namanya setTimeout() yg setTimeout() ini menerima parameter berupa callback atau kalian inget kita pernah belajar mengenai Higher Order function ya, bahwa function bisa menerima parameter berupa function lagi dan itu disebut dengan Callback.

Nah Callback ada tipe nya Asynchronous contoh nya setTimeout() itu, tapi gausah bingung dulu ya pokoknya proses yg kompleks tadi kita masukan kedalam setTimeout(), atau kalo mau disimulasikan tidak menggunakan function prosesYangKompleks() nya kita bisa simulasikan gini deh :

        console.log('satu');

        setTimeout(() => {

            console.log('dua');

        }, 5000);

        console.log('tiga');

Jadi ada sebuah perintah sederhana tampilin console.log(‘dua’) disimpan kedalam setTimeout() sebagai Callbak yang parameter delay nya atau nunggu nya selama 5000milisecond atau 5detik, jadi maksudnya gini pertama tampilin dulu console.log(‘satu’) terus tunggu 5detik tampilin console.log(‘dua’) terus tampilin console.log(‘tiga’).

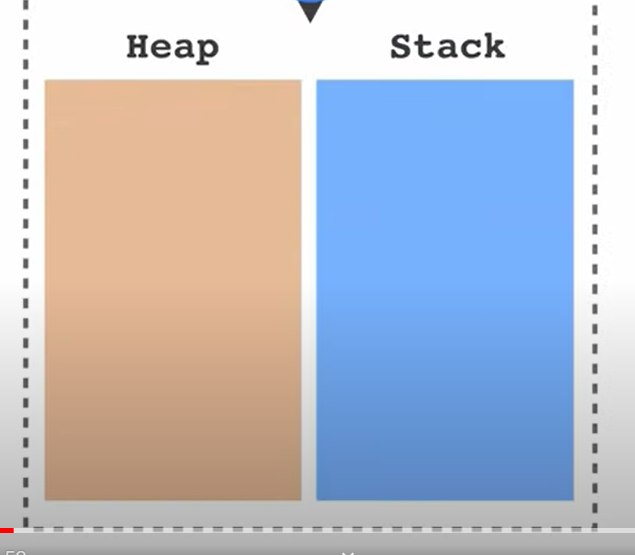
Nah ini kalo kita jalanin jadinya gini pertamkan pastikan dibaca dulu console.log(‘satu’) ya masuk kedalam stack eksekusi selesai hilang turun masuk ke setTimeout() jalanin masuk ke STACK nya nah itukan harusnya kita nunggu 5detik ya tapi ternyata setTimeout() ini langsung hilang dari Stack nya padahal belum munculin ‘dua’ ke console nya, dan programnya langsung ngebaca ke console.log(‘tiga’) lalu selesai di eksekusi hilang dari Stack dan tiba-tiba setelah 5detik nunggu console.log(‘dua’) dijalankan dan tampil ke console.

Nah jadi kalo menggunakan teknik itu yg tampil adalah ‘satu’ terus ‘tiga’ dan baru terakhir ‘dua’ dan ‘dua’ itu tampil setelah nunggu 5detik jadi setTimeout() itu awal nya udah masuk ke STACK lalu dia hilang dari stack tapi belum ngejalanin apa-apa yg ada didalam setTimeout() itu tapi dia ngejalanin code yg ada diluar itu dulu kalo udah selesai semua balik lagi ke setTimeout() nunggu 5detik tapi ini yg masuk ke STACK nya si console.log(‘dua’) bukan setTimeout() nya karena dia udah sebelum nya gitu, jadi kaya setelah ngejalanin console.log(‘tiga’) kan dibawah nya ceritanya gaada program lagi jadi dia ngelanjutin setTimeout() yg tadi belum selesai dijalanin gitu.

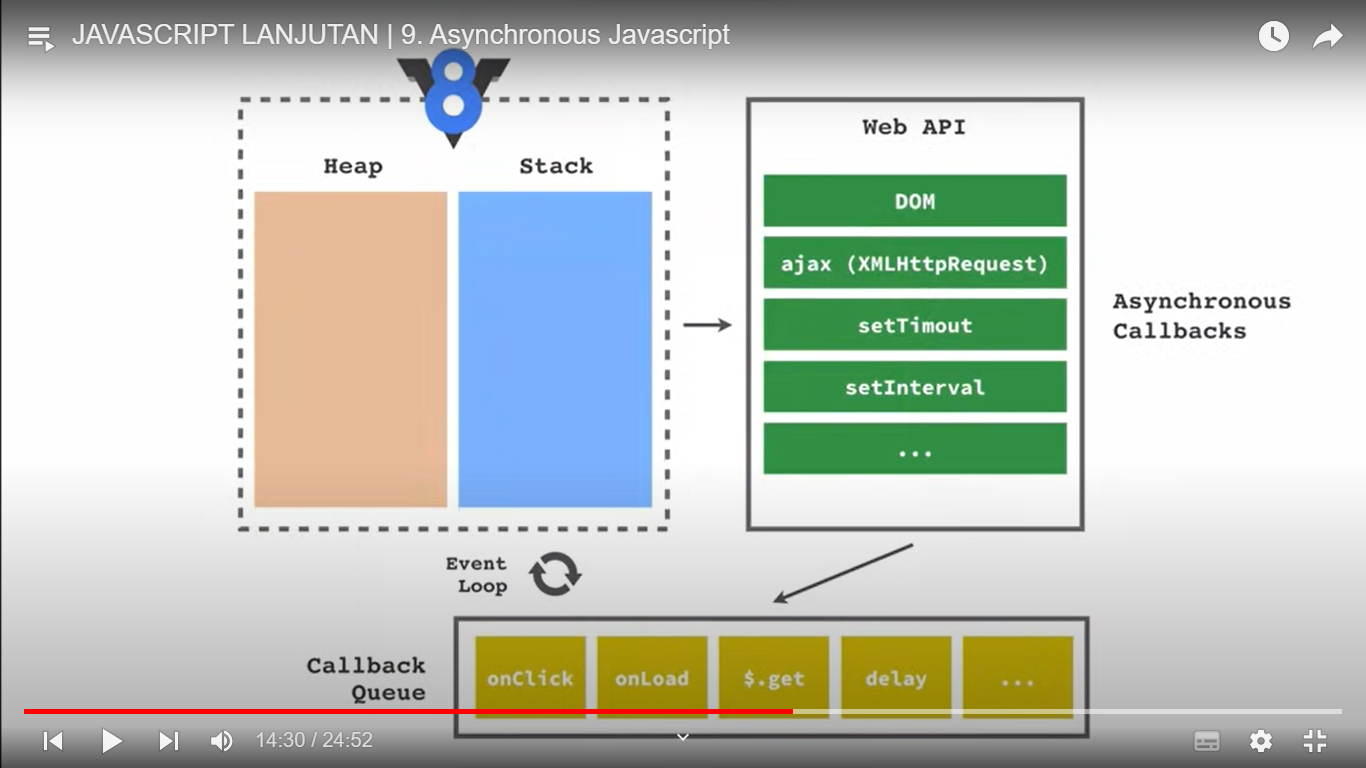
Kembali lagi jadi console.log(‘tiga’) itu ga keblok kode kalian bisa tetep jalan sampai bawah dan console.log(‘dua’) itu akan tampil setelah nunggu 5detik lalu kalo udah selesai di eksekusi tentu saja dia hilang dari STACK.

**Asynchronous Callback dan Event loop**

Nah kenapa bisa begitu, jadi itu tadi yg saya sebut dengan menggunakan Asynchronous Callback dan itu bisa terjadi karena ada seusuatu yg namanya Event loop, nah apalagi nih Event Loop coba kita balik lagi ke arsitektur dari V8 kita ya.



Jadi V8 Cuma ada 2 ada HEAD dan STACK nah sebetulnya V8 itu buat ngejalanin javacript nya kan, nah sebetul nya arsitektur lengkap nya seperti ini:



Seperti itu kalo kalian jalanin nya didalam browser jadi V8 itu bagian dari arsitektur browser kalian, ini pak dhika jelasin sedikit ya, tadi mungkin agak aneh kenapa setTimeout() bisa hialng ya.

Nah sebenernya setTimeout() ngga hilang jadi kalo ketemu setTimeout() sebenerna perintah yg ada didalam setTimeout() itu dipindahkan ke sesuatu yg namanya Web API, jadi web API itu menangani Callback yg Asynchronous yg bukan merupakan dari V8.

Contoh nya kalo kalian punya pengelolaan DOM misalkan saat tombol di klik saat mouse digerakan kalian punya eventListener itu dikelola oleh WEB API, terus kalo kalian punya proses AJAX itu juga di kelola oleh web API mau manual menggunakan XMLHttpRequest atau menggunakan JQUERY misalnya itu ditanganin oleh web API.

Jadi kalo ada di STACK di pindah dulu ke web API begitu pula dengan setTimeout(), setInterval() atau perintah lain yg merupakan Asynchronous Callback yg juga bagian dari web API, gitu ya jadi kenapa hilang gitu aja karena masuk ke web API.

Nah didalam web API setelah waktu tunggu nya selesai atau DOM setelah tombol nya diklik atau kalo ajax setelah data berhasil diambil dari API tujuan nah setelah selesai harusnya callback dijalankan ya, nah tapi web API itu gabisa langsung menjalankan kembali fungsi nya kedalam STACK.

Nah callback ya harus disimpan dulu ke sebuah tempat yg namanya Callback Queue jadi ada sebuah tempat lagi nih buat nyimpen antrian dari callback nya, jadi kalo tombol diklik waktunya sudah selesai Interval nya sudah masuk pindah dia ke Callback Queue.

Nah setelah ada di callback queue maka ada sesuatu lagi nih yg namanya Event Loop, yg tugas dia simpel aja dia ngecek di callback queue ada antrian ga kalo ad aambil callback nya simpan kedalam Stack, nah kalo ada di stack karena dia udah punya si V8 itu yaudah jalanin seperti jalannya V8 biasa.

Tapi syarat nya kapan sebuah callback bisa masuk kedalam stack ketika didalam stack nya sudah tidak ada tumpukan perintah yg akan di eksekusi jadi harus kosong dulu didalam stack nya baru event loop bisa masukin callback queue nya kedalam stack gitu.

Jadi kalo pake callback gini code programnya akan dijalankan seluruh nya setelah semua kodingan yg diluar fungsi yg tadi misal setTimeout() itu udah dijalankan baru deh kode yg didalam fungsi setTimeout() itu bisa dijalankan di Stack gitu.

Atau kita coba balik lagi ke kasus kita ke codingan yg ini :

        console.log('satu');

        setTimeout(() => {

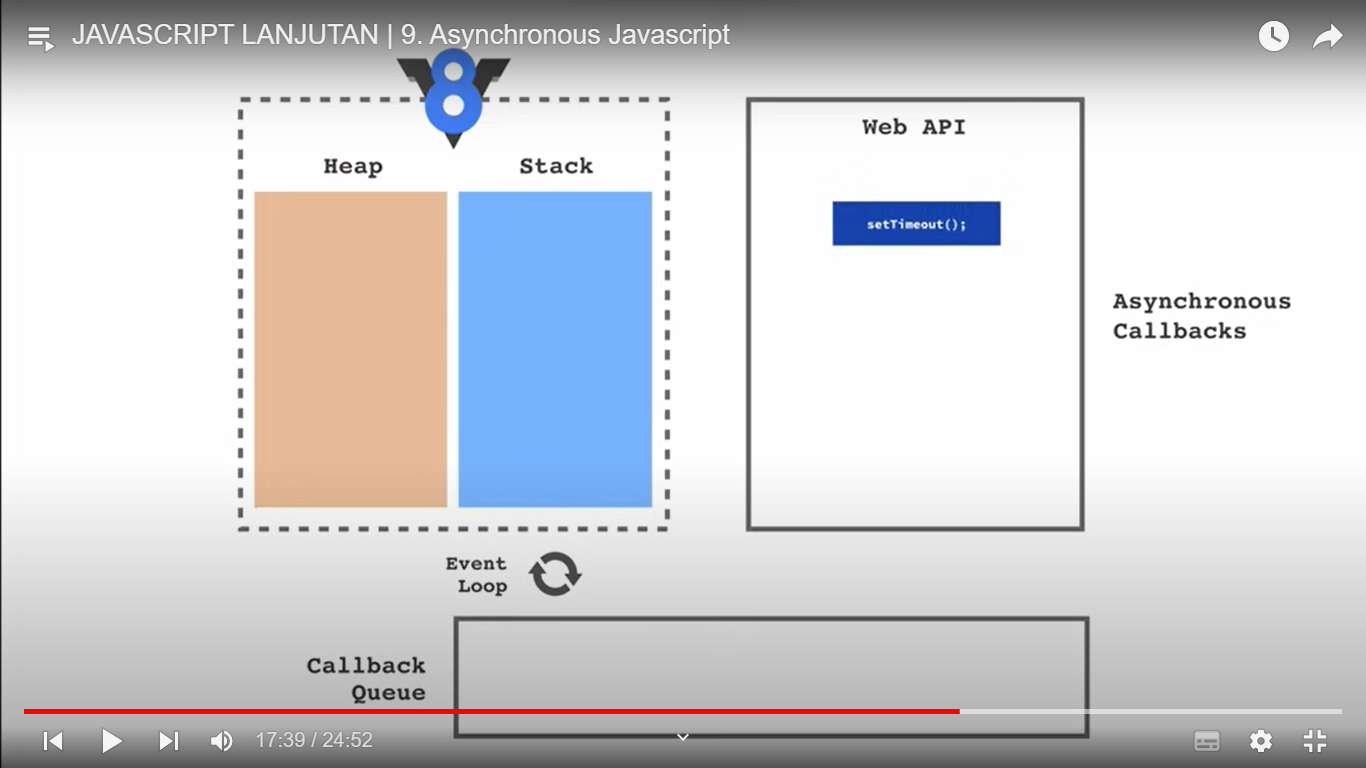
            console.log('dua');

        }, 5000);

        console.log('tiga');

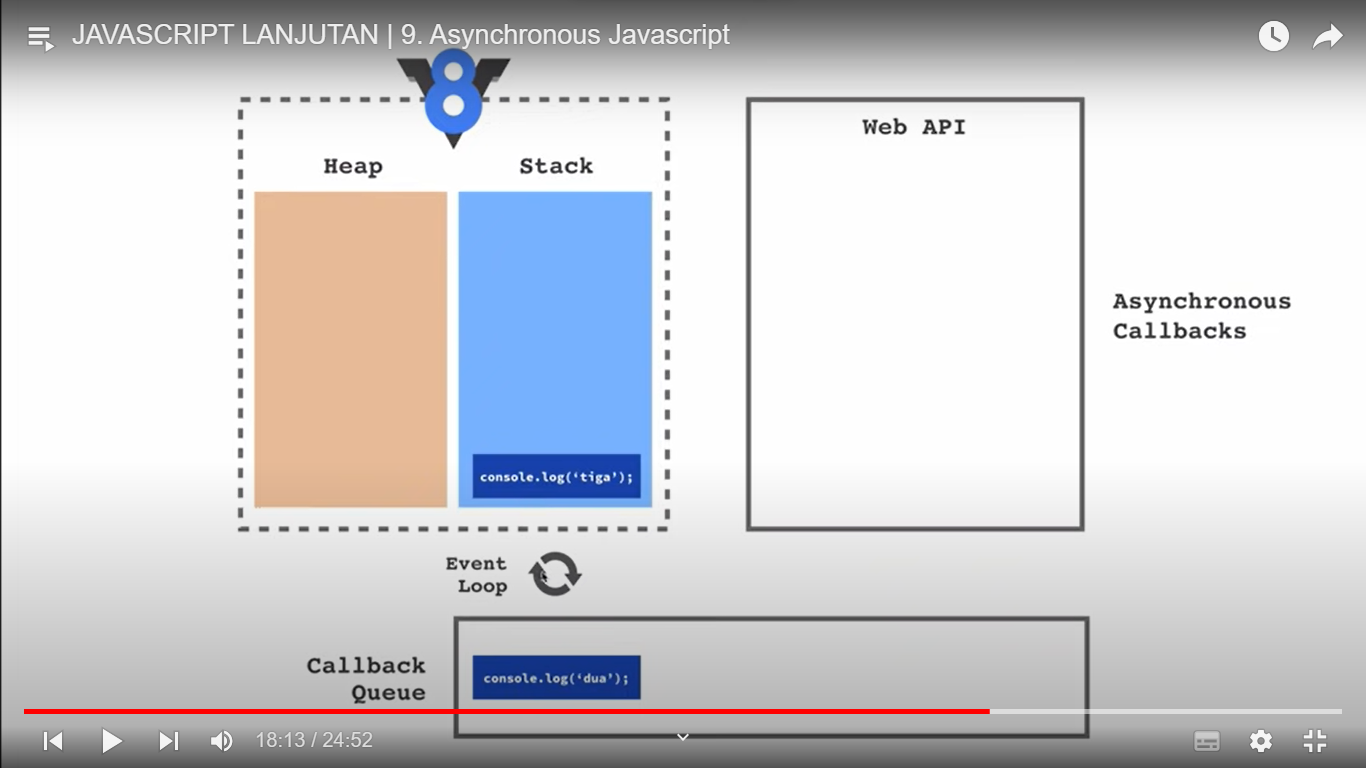
Coba kita simulasikan kedalam arsitektur kita, yg pertama console.log(‘satu’) dulu, sebetul nya console.log(‘satu’) itu bagian dari WEB API bukan punya nya V8 tapi karena console.log(‘satu’) itu bukan Asynchronous Callbacks maka bisa dijalanin di Stack jadi udah dijalanin selesai tampil hilang dari stack nya.

Setelah itu masuk ke setTimeout() nah dia Asynchronous Callbacks punya web API maka pindahin ke situ dan jadi nya hilang deh dari Stack.



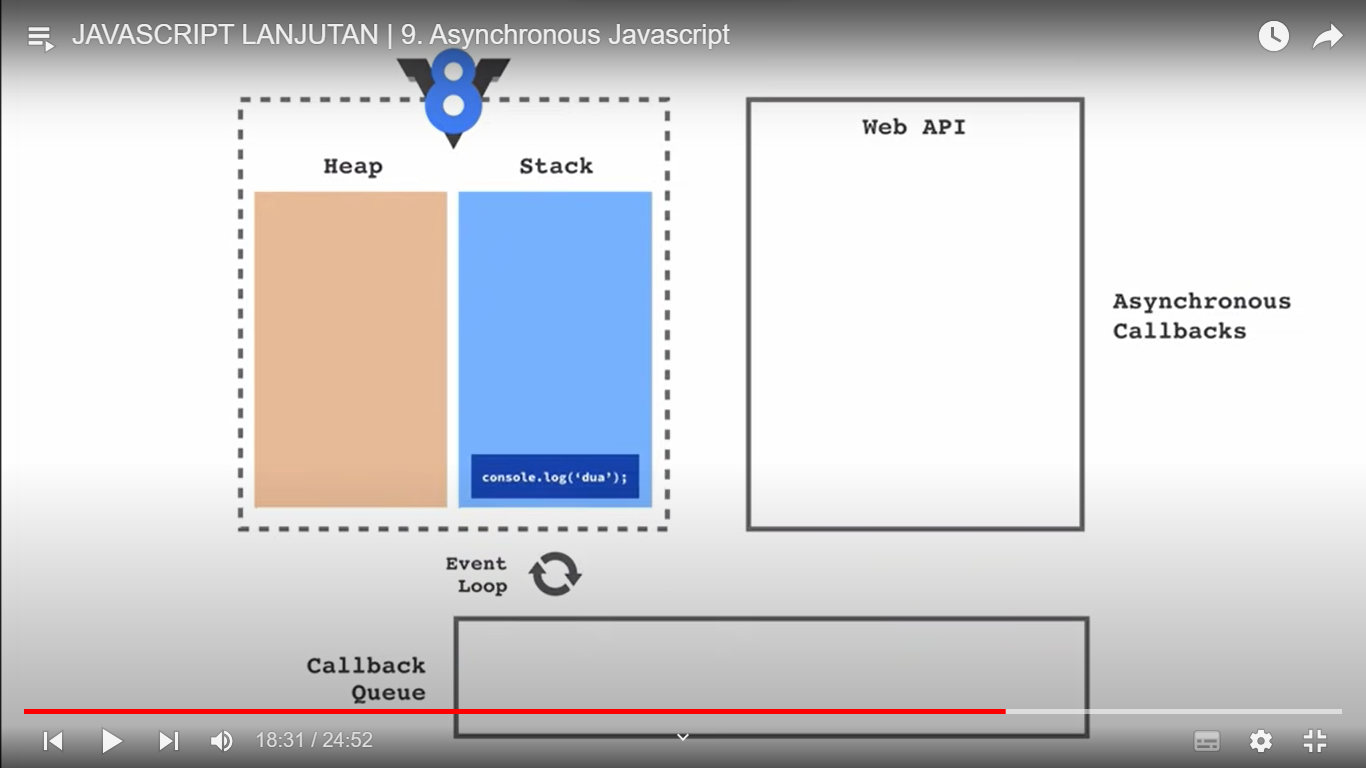
Nah setelah dipindahin kan hilang dari stack jadi kita jalanin codingan selanjutnya nah console.log(‘tiga’) masuk ke stack, nah sekarang kondisi di web API nya dia nunggu dulu 5detik ya sampai callback nya siap di eksekusi tapi STACK nya terus mengeksekusi apa yg ada didalam program nya.

Jadi setelah 5detik setTimeout() siap di eksekusi sebetulnya nah tapi kan dair web API itu kan gabisa dia langsung masuk ke STACK nya dia harus pindah dulu ke callback queue atau antrian callback nya, gini :



Nah kalo udah disitu Event loop akan ngecek ada callback ga di antrian nya ohh ada ternyata tapi callback itu gabisa dimasukan ke STACK karena didalam stack nya masih ada perintah yg console.log(‘tiga’) itu.

Jadi perintah nya harus beres dulu dan dia keluar dari stack jadi stack nya kosong baru deh sekarang event loop nya akan masukin ke stack nya, gini :

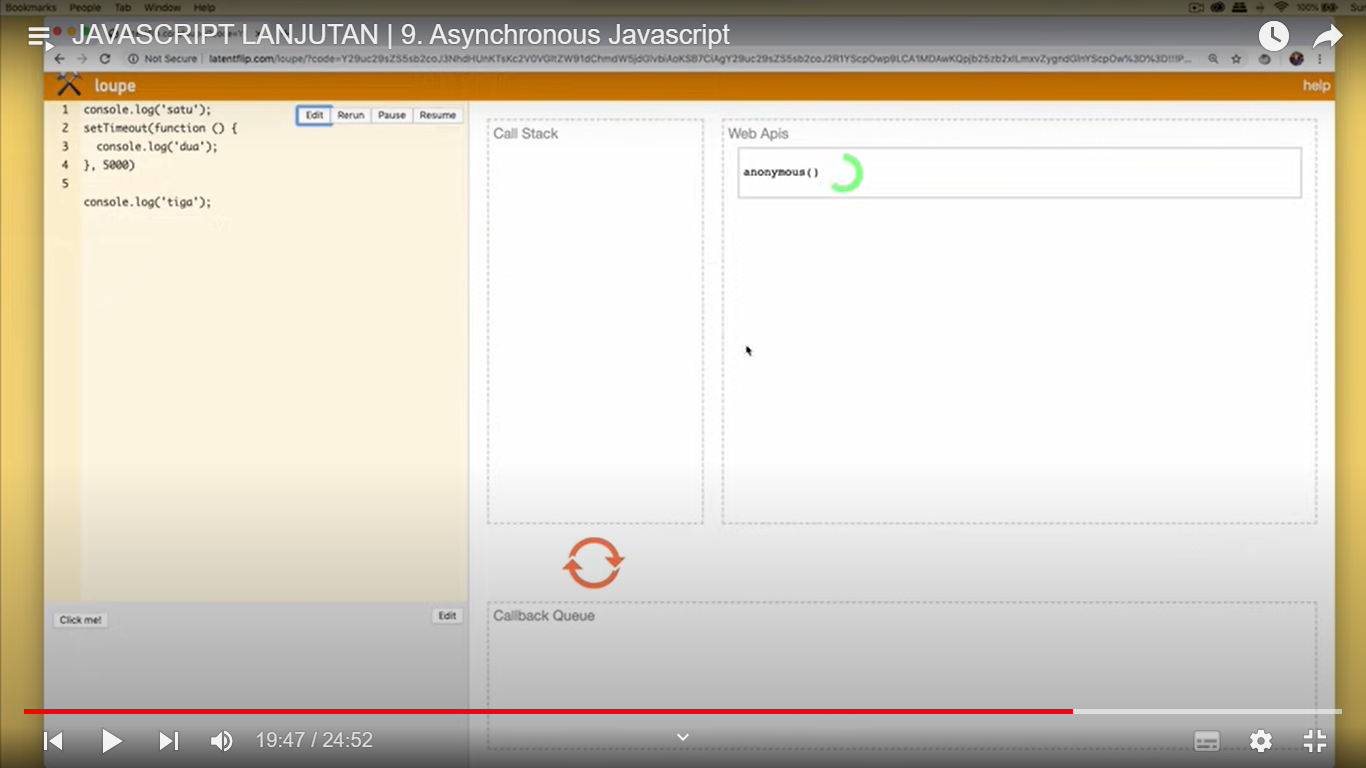


Makannya itu yg tampil di console ya ‘satu’ terus ‘tiga’ baru deh ‘dua’ karena aristektur itu nah kalo udah di stack setTimeout() nya tinggal di eksekusi aja tampil terus hilang dari stack deh.

Atau kalo misalkan kita langsung cobain aja yg ada didalam websaite ini ya :

<http://latentflip.com/loupe/> by. Phillip Roberts

nanti muncul ada video youtube nya kalo kalian mau nonton atau kalian mau langsung tools yg dia buat di web nya, jadi web nya itu meilustrasikan bagaimana kode program kita dijalankan coba kunjungi deh, kaya gini :



Nah jadi dia memperlihat kan codingan kita yg kasus tadi dengan jelas proses nya seperti apa, bisa dicoba langsung. Kalo proses nya kecepetan kalian mau lihat lebih lambat proses perpindahan kode nya kalian bisa ke setting nya yg pojok atas kiri nanti ada delay nah tinggal atur aja untuk seolah” animasi nya lebih lama gitu.

Nah itu tools nya jadul jadi belum bisa menerima Arrow function jadi harus pake function biasa, jadi tools nya untuk mensimulasikan proses yg lama, nah tapi ini aneh nya gini kalo kita ganti waktu tunggu di setTimeout() nya yg tadi nya 5detik jadi 0 harusnya kan gaada waktu tunggu ya.

Nah meskipun gaada waktu tunggu karena setTimeout() itu Asynchrounous function punya nya web API dia tetep masuk dulu ke web API nya padahal harusnya kalo gaada waktu tunggu yaudah tampil nya berurutan ‘satu’, ‘dua’, ‘tiga’ gitu ya.

Nah tapi karena dia Asynchrounous callback punya nya web API tetep harus pindah dulu ke situ ya walaupun langsung cepet pindah nya ke Callback antrian nya harus nya langsung aja abis ke callback Queue nya di event loop ke stack, tapi itu gabisa karena even loop nya nunggu dulu stack nya harus kosong.

Jadi sebenernya kalo ngakalin kita bikin sebuah Asynchrounous function kita bisa bikin waktu tunggu nya itu 0 gitu.

CONTOH

Ini contoh dari websaite nya langsung ya :

        $.on('button', 'click', function onClick() {

            setTimeout(function timer() {

                console.log('You clicked the button!');

            }, 2000);

        });

        console.log("Hi!");

        setTimeout(function timeout() {

            console.log("Click the button!");

        }, 5000);

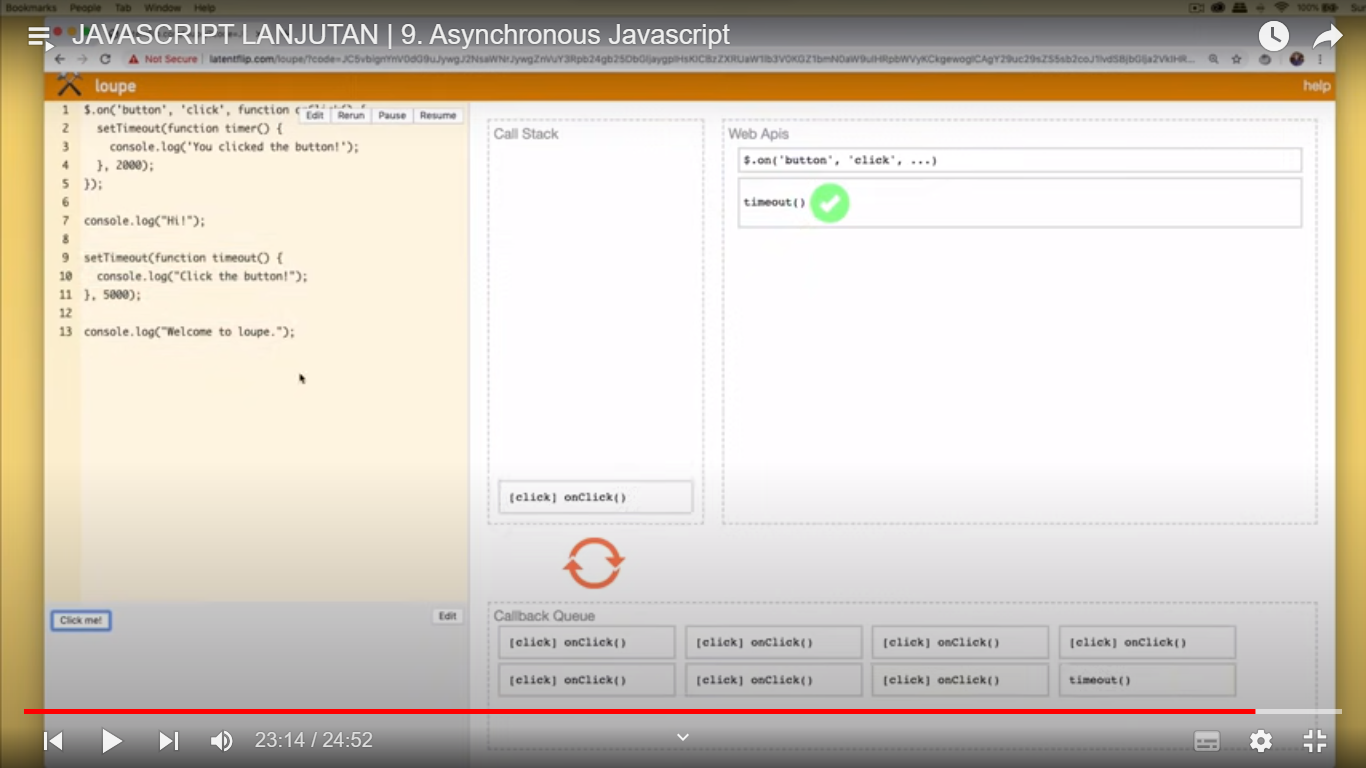
        console.log('Welcome to loupe');

Jadi yg pertama ada EventHandler itu pake jquery ya, pokoknya intinya ketika tombol di klik jalankan callback berikut callback nya dikasih nama onClick sebetulnya gausah pun gapapa tapi nanti biar disitu ada namanya.

Nah tombol ada kita harus bikin html nya, jadi kalo mau langsung nyoba kunjungi websaite nya aja biar langsung nyoba, nah setelah itu ada console.log(‘Hi!’) itu biasa aja bukan Asynchrounous callback, lalu ada lagi setTimeout() callback dan terakhir ada lagi console.log.

Jadi kalo dijalanin yg pertama callback dia nunggu di web API nya nunggu diklik tombol nya baru jalan terus masuk ke console.log(‘Hi!’), terus ke setTimeout() berikutnya masuk ke web API nunggu 5detik masuk ke console.log(‘Welcome to loupe’) dijalanin, terus sekarang harusnya fungsi timeout() nya masuk ke bawah kalo udah selesai masuk ke Queue di cek udah kosong terus event loop nya jalanin.

Nah tapi callback yg baris awal itu nunggu terus di web API karena tombolnya ga diklik-klik, coba kita ulangin tapi sekarang saya klik tombol nya, kita run terus callbacknya masuk ke web API dulu lalu kita klik dia langsung masuk ke QUEUE terus jalan kode selanjutnya console.log(‘Hi!’) terus setTimeout() masuk ke stack lalu ke web API tunggu 5detik terus dia masuk ke antrian callback nya ke QUEUE itu, terus jalanin ada lagi console.log(‘Welcome to loupe’) baru deh kalo udah hilang dari STACK dan stack nya udah kosong Event loop nya mindahin yg ada di antrian Callback QUEUE, gini kalo antrian nya :



Nah itu tombol nya diklik lebih dari satu kali ya, jadi begitu perilaku yg ada didalam stack nih. Jadi intinya kita pelajarin itu semau supaya kita bisa menciptakan sebuah program yg nyaman digunakan oleh user, jadi kita tau kapan menggunakan fungsi ASYNCHROUNOUS pada javascript, jadi intinya sebetul nya jangan sampe kita block si event loop nya.

Misal kalian bikin aplikasi yg ngeleg mungkin aja karena event loop nya itu terlalu menumpuk dan kita akan mempelajari bagaimana cara mensiasatinyadi video berikutnya.