NAMA : NADHEA DENI PUTRI
NIM : 120140128
PENGEMBANGAN APLIKASI MOBILE
TUGAS INDIVIDU 2

RANGKUMAN MATERI JAVASCRIPT LANJUTAN

1. **Closure**

   Closure adalah fungsi yang mengingat variabel luarnya dan dapat mengaksesnya. Dalam beberapa bahasa, itu tidak mungkin, atau fungsi harus ditulis dengan cara khusus untuk mewujudkannya. Tetapi seperti yang dijelaskan di atas, dalam JavaScript, semua fungsi secara alami adalah penutupan (hanya ada satu pengecualian, untuk dicakup dalam sintaks "Fungsi baru"). Artinya, mereka secara otomatis mengingat di mana mereka dibuat menggunakan properti yang tersembunyi , dan kemudian kode mereka dapat mengakses variabel luar.

   Fungsi

   - Sebagai enkapsulasi data, closure berguna untuk membuat variabel privat dan fungsi privat. Dengan cara ini, penutupan dapat menyimpan data dalam rentang terpisah dan hanya mengakses/menggunakannya saat diperlukan.
   - Membuat fungsi dinamis, kita dapat mengatur komposisi fungsi dan kode yang dapat digunakan kembali.

2. **Immediately Invoked Function Expression**

3. **First-class function**

   pemrograman dikatakan memiliki fungsi kelas satu jika fungsi dalam bahasa tersebut diperlakukan seperti variabel lainnya. Jadi fungsi dapat ditugaskan ke variabel lain atau diteruskan sebagai argumen atau dapat dikembalikan oleh fungsi lain. JavaScript memperlakukan fungsi sebagai warga negara kelas satu. Ini berarti bahwa fungsi hanyalah sebuah nilai dan hanyalah jenis objek lain.

4. **Higher-order function**

   Fungsi tingkat tinggi adalah fungsi yang beroperasi pada fungsi lain, baik dengan menganggapnya sebagai argumen atau dengan mengembalikannya. Dengan kata sederhana, fungsi Higher-Order adalah fungsi yang menerima fungsi sebagai argumen atau mengembalikan fungsi sebagai output.

5. **Execution Context**

   Sederhananya, Execution Context adalah konsep abstrak dari lingkungan di mana kode Javascript dievaluasi dan dieksekusi. Setiap kali kode apa pun dijalankan di JavaScript, itu dijalankan di dalam Execution Context.

Jenis Execution Context

Ada tiga jenis Execution Context dalam JavaScript.

1. **Execution Context Global**

   Ini adalah Execution Context default atau dasar. Kode yang tidak ada di dalam fungsi apa pun berada dalam Execution Context global. Ini melakukan dua hal: ia menciptakan objek global yang merupakan objek jendela (dalam kasus browser) dan menetapkan nilai `this` sama dengan objek global. Hanya ada satu Execution Context global dalam suatu program.

2. **Execution Context Fungsional**

   Setiap kali fungsi dipanggil, Execution Context baru dibuat untuk fungsi tersebut. Setiap fungsi memiliki Execution Contextnya sendiri, tetapi dibuat ketika fungsi invoked atau called. Mungkin ada sejumlah Execution Context fungsi. Setiap kali Execution Context baru dibuat, ia melewati serangkaian langkah dalam urutan yang ditentukan yang akan saya bahas nanti di artikel ini.

3. **Execution Context Fungsi Eval**

   Kode yang dijalankan di dalam fungsi eval juga mendapatkan Execution Contextnya sendiri, tetapi karena `eval` biasanya tidak digunakan oleh pengembang JavaScript, jadi saya tidak akan membahasnya di sini.

6. Execution Stack

Execution stack, juga dikenal sebagai " calling stack" dalam bahasa pemrograman lain, adalah stack dengan struktur LIFO (Last in, First out), yang digunakan untuk menyimpan semua Execution Context yang dibuat selama eksekusi kode.

Ketika mesin JavaScript pertama kali menemukan skrip Anda, ia menciptakan Execution Context global dan mendorongnya ke Execution Stacksaat ini. Setiap kali mesin menemukan pemanggilan fungsi, itu menciptakan Execution Context baru untuk fungsi itu dan mendorongnya ke atas stack.

Mesin menjalankan fungsi yang Execution Contextnya berada di bagian atas stack. Ketika fungsi ini selesai, Stack eksekusinya muncul dari stack, dan kontrol mencapai konteks di bawahnya dalam stack saat ini.

Ketika kode di atas dimuat di browser, mesin Javascript membuat Execution Context global dan mendorongnya ke Execution Stacksaat ini. Ketika panggilan ke `first()` ditemukan, mesin Javascript membuat Execution Context baru untuk fungsi itu dan mendorongnya ke bagian atas Execution Stacksaat ini.

Ketika fungsi `second()` dipanggil dari dalam fungsi `first()`, mesin Javascript membuat Execution Context baru untuk fungsi tersebut dan mendorongnya ke bagian atas Execution Stacksaat ini. Ketika fungsi `second()` selesai, konteks eksekusinya muncul dari stack saat ini, dan kontrol mencapai Execution Context di bawahnya, yaitu Execution Context fungsi `first()`.

Ketika first() selesai, stack eksekusinya dihapus dari stack dan kontrol mencapai Execution Context global. Setelah semua kode dijalankan, mesin JavaScript menghapus Execution Context global dari stack saat ini.

7. **Event Loop**

Event loop adalah rahasia di balik pemrograman async JavaScript. JS menjalankan semua operasi pada satu utas, tetapi menggunakan beberapa struktur data pintar, ini memberi kita ilusi multi-threading. Mari kita lihat apa yang terjadi di back-end. call stack bertanggung jawab untuk melacak semua operasi yang sejalan untuk dieksekusi. Setiap kali fungsi selesai, fungsi akan muncul dari stack. The event queue bertanggung jawab untuk mengirim fungsi baru ke stack untuk diproses. Ini mengikuti struktur data antrian untuk mempertahankan urutan yang benar di mana semua operasi harus dikirim untuk dieksekusi.

Setiap kali fungsi async dipanggil, fungsi tersebut dikirim ke API browser. Ini adalah API yang dibangun ke dalam browser. Berdasarkan perintah yang diterima dari stack call, API memulai operasi single-threaded-nya sendiri.

Contohnya adalah metode `setTimeout`. Ketika operasi `setTimeout` diproses dalam stack, itu dikirim ke API yang sesuai yang menunggu sampai waktu yang ditentukan untuk mengirim operasi ini kembali untuk diproses di The event queue. Oleh karena itu, digunakan cyclic system untuk menjalankan operasi async di JavaScript. Bahasa itu sendiri berulir tunggal, tetapi API browser bertindak sebagai utas terpisah.

8. **Callbacks**

Callback adalah sebuah fungsi yang dikirimkan sebagai parameter fungsi lainnya. Fungsi diatas adalah sebuah contoh callback yang berjalan secara Synchronous karena fungsi callback langsung dieksekusi pada sebuah proses fungsi yang memiliki sifat synchronous.

Callback memastikan bahwa suatu fungsi tidak akan berjalan sebelum tugas selesai tetapi akan berjalan tepat setelah tugas selesai. Ini membantu kami mengembangkan kode JavaScript async dan membuat kami aman dari masalah dan kesalahan.

Dalam JavaScript, cara untuk membuat fungsi callback adalah dengan meneruskannya sebagai parameter ke fungsi lain, dan kemudian memanggilnya kembali tepat setelah sesuatu terjadi atau beberapa tugas selesai.

Fungsi dalam javascript adalah object atau sering disebut *first-class object*, yang artinya :

- Function bisa di jadikan parameter
- Function dapat disimpan ke dalam variabel
- Seperti object pada umumnya, function bisa memiliki property dan method
- Function dapat mengembalikan value dalam bentuk function

Callback dapat digunakan untuk proses synchronous maupun asynchronous. Seperti, Injeksi atau modifikasi hasil eksekusi sebuah function, Event listener, hingga menangani proses asynchronous.

9. **Promises dan Async/Await**

Promises:

Promise di NodeJS mirip dengan promise di kehidupan nyata. Ini adalah jaminan bahwa sesuatu akan dilakukan. Promise digunakan untuk melacak apakah peristiwa async telah dijalankan atau tidak dan menentukan apa yang terjadi setelah peristiwa terjadi. Ini adalah objek yang memiliki 3 keadaan yaitu:

1. **Pending:** Status Awal, sebelum peristiwa itu terjadi.
2. **Resolved:** Setelah operasi selesai dengan sukses.
3. **Rejected::** Jika operasi memiliki kesalahan selama eksekusi, promise gagal.

Async/Await:

Async/Await digunakan untuk bekerja dengan promise dalam fungsi async. Ini pada dasarnya adalah gula sintaksis untuk promise. Ini hanya pembungkus untuk mengubah gaya kode dan membuat promise lebih mudah dibaca dan digunakan. Itu membuat kode async lebih mirip kode sinkron / prosedural, yang lebih mudah dipahami.

await hanya dapat digunakan dalam fungsi async. Ini digunakan untuk memanggil fungsi async dan menunggunya untuk menyelesaikan atau menolak. await memblokir eksekusi kode dalam fungsi async di mana ia berada.

Perbedaan Antara Promise dan Async/Await:

no	Promises	Async/Await
1.	Promise adalah objek yang mewakili keadaan operasi menengah yang dijamin untuk menyelesaikan eksekusinya di beberapa titik di masa depan.	Async/Await adalah gula sintaksis untuk promise, pembungkus yang membuat kode dijalankan lebih sinkron.
2.	Promise memiliki 3 negara bagian terselesaikan, ditolak, dan tertunda.	Itu tidak memiliki negara bagian. Ini mengembalikan promise baik diselesaikan atau ditolak.
3.	Jika fungsi "fxn1" adalah untuk mengeksekusi setelah promise, maka promise.then(fxn1) melanjutkan eksekusi fungsi saat ini setelah menambahkan panggilan fxn1 ke rantai callback.	Jika fungsi "fxn1" adalah untuk mengeksekusi setelah menunggu, maka await X() menangguhkan eksekusi fungsi saat ini dan kemudian fxn1 dieksekusi.
4.	Penanganan kesalahan dilakukan dengan menggunakan metode .then() dan .catch().	Penanganan kesalahan dilakukan dengan menggunakan metode .try() dan .catch().
5.	Rantai promise terkadang menjadi sulit untuk dipahami.	Menggunakan Async/Await membuatnya lebih mudah untuk membaca dan memahami alur program dibandingkan dengan rantai promise.

Github: https://github.com/Nadhea997/Individu-2