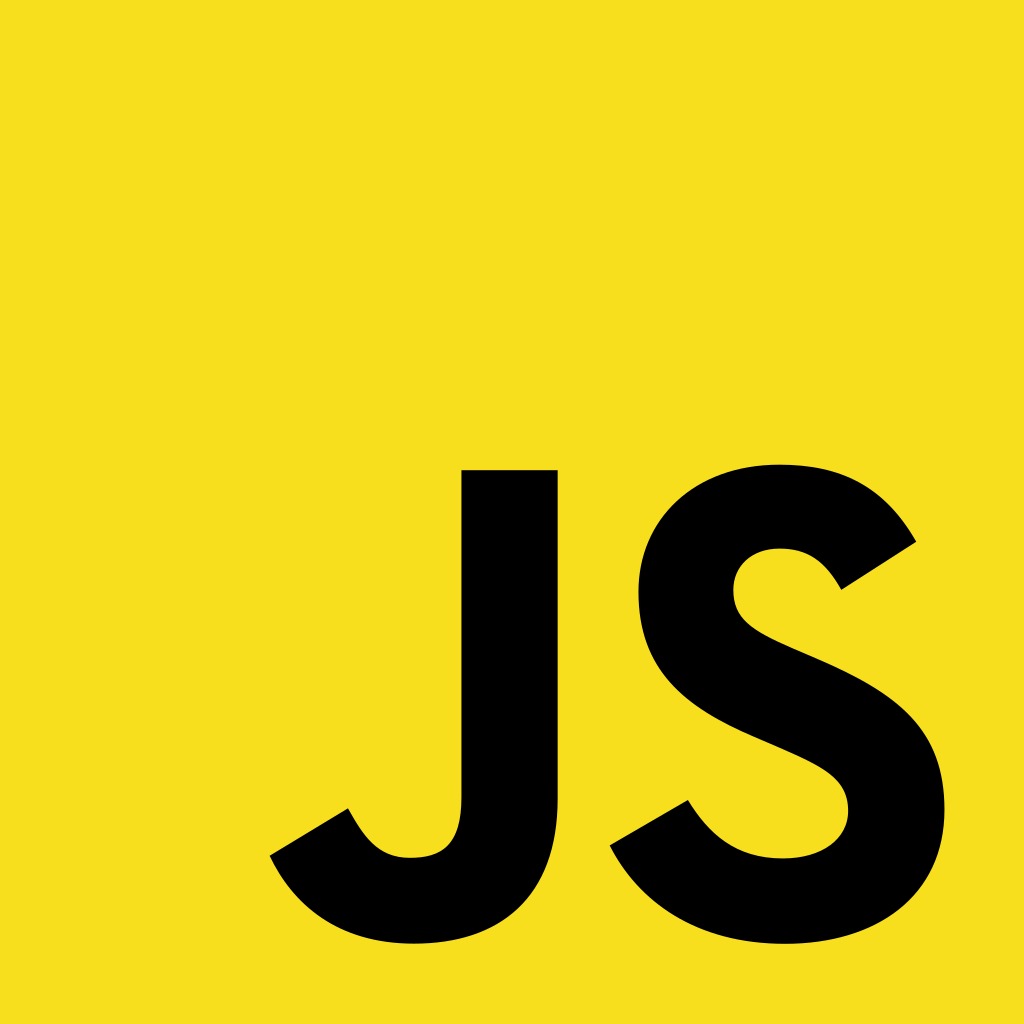
**[Belajar Dasar Pemrograman JavaScript](https://www.dicoding.com/academies/256/corridor)**

|  |
| --- |
| *Modul kelas Belajar Dasar Pemrograman JavaScript dalam bentuk cetak (buku) maupun elektronik sudah didaftarkan ke Dirjen HKI, Kemenkumham RI. Segala bentuk penggandaan dan atau komersialisasi, sebagian atau seluruh bagian, baik cetak maupun elektronik terhadap modul kelas Belajar Dasar Pemrograman JavaScript tanpa izin formal tertulis kepada pemilik hak cipta akan diproses melalui jalur hukum.  Hak cipta dilindungi oleh Undang-undang © Dicoding 2017 - 2023.* |

Dengan lanjut ke materi berikutnya, Anda menyetujui adanya informasi hak cipta di atas terhadap modul dalam kelas ini.

**Pengenalan Kelas**



Selamat datang di kelas Belajar Dasar Pemrograman JavaScript!

Pada kelas ini kita akan belajar hal-hal fundamental yang perlu Anda ketahui tentang bahasa pemrograman JavaScript pada lingkungan Node JS. Untuk penggunaan JavaScript pada lingkungan browser, Anda dapat mempelajarinya pada kelas [Belajar Dasar Pemrograman Web](https://www.dicoding.com/academies/123).

Materi yang tersaji ramah untuk pemula, sehingga tidak ada prasyarat pemrograman yang perlu Anda kuasai untuk belajar di kelas ini.

Beberapa hal yang akan dipelajari pada kelas ini, antara lain:

* **Introduction to JavaScript** : Mengenal apa itu JavaScript, mulai dari sejarah hingga alasan untuk mulai mempelajarinya.
* **JavaScript Fundamentals** : Materi fundamental terkait JavaScript, seperti *statement*, *expressions*, variabel, tipe data, dan *control flow*.
* **Data Structure** : Menjelaskan bagaimana menyimpan dan mengelola data yang lebih kompleks menggunakan Object, Array, Map, Set, dsb.
* **Function** : Mempelajari bagaimana mengelompokkan kode menjadi fungsi yang dapat dipanggil ulang.
* **Object-Oriented Programming** : Memahami konsep OOP pada JavaScript, bagaimana membuat class, properti, dan method dari class.
* **Functional Programming** : Memahami konsep *functional programming* seperti perilaku fungsi, *anonymous function*, dan *higher-order*.
* **Setup Environment** : Proses instalasi lingkungan pengembangan seperti Node JS dan IDE pada perangkat yang Anda miliki.
* **Module** : Mempelajari bagaimana beberapa berkas JavaScript dapat saling berkomunikasi dan berbagi data satu sama lain.
* **Error Handling** : Mempelajari bagaimana menangani kegagalan yang mungkin muncul dalam program, sehingga tidak berdampak kepada pengguna.
* **Concurrency** : Mempelajari bagaimana menangani dan membuat proses *asynchronous* pada JavaScript.
* **Node Package Manager** : Mengetahui beberapa *library* yang penting untuk ditambahkan ke dalam proyek aplikasi JavaScript.
* **JavaScript Testing** : Mempelajari bagaimana melakukan pengujian otomatis pada kode JavaScript.

Setelah mengetahui materi apa saja yang akan dipelajari, mari kita mulai melangkah ke materi pertama yaitu Pengenalan JavaScript. *Let’s go!*

## Mekanisme Belajar

Selamat datang di Dicoding Academy. Sebelum memulai belajar di kelas ini, Anda perlu tahu tahapan dan cara belajar beserta fasilitas yang tersedia agar proses belajar lebih efektif.

### Materi Pembelajaran

* **Materi Bacaan Elektronik**  
  Materi yang ada dalam kelas ini mayoritas berupa teks atau tulisan. Mengapa demikian? Karena kami menemukan bahwa dalam proses belajar di bidang pemrograman, bentuk materi pembelajaran yang paling efektif untuk diingat, dimengerti, dan yang terpenting, dipraktikkan, adalah dalam bentuk teks.
* **Forum Diskusi**  
  Setiap kelas memiliki sebuah forum diskusi yang dapat Anda gunakan untuk bertanya (dan menjawab) mengenai materi kelas. Instruktur kami yang terdiri dari Curriculum Developer dan Academy Code Reviewer siap membantu Anda melalui forum diskusi ini. Tak hanya kami, Anda pun dapat berpartisipasi. Untuk meningkatkan retensi ilmu yang Anda punya, berbagi adalah salah satu kuncinya. Jadi, silakan aktif dan saling membantu di dalam forum.  
    
  Jika Anda menemui kendala perihal situs Dicoding dan administrasi kelas, tanyakan kepada kami melalui tombol chat di kanan bawah layar Anda ketika membuka situs Dicoding. Jangan gunakan forum diskusi ya karena itu salah alamat.

### Evaluasi Pembelajaran

* **Ujian (Kuis, Ujian Akhir Kelas, Knowledge Check)**  
  Terdapat berbagai jenis ujian di kelas guna mengecek pemahaman Anda pada materi pembelajaran. Ketersediaan ujian beserta jenis ujian yang tersedia pada setiap kelas dapat berbeda-beda. Setiap pertanyaan dalam ujian pasti mencakup materi yang telah dibahas. Sehingga jika ada pertanyaan yang tidak dapat Anda jawab, pastikan Anda mengulang kembali pembelajaran.

## Forum Diskusi

### Ekspektasi

Dengan banyaknya jumlah siswa Dicoding Academy, kami tidak mengharapkan siswa untuk membaca semua diskusi atau komentar pada forum diskusi. Sebaliknya, bacalah hal-hal yang menurut Anda menarik dan dapat membantu Anda dalam menyelesaikan kelas. Lebih baik lagi, jika Anda dapat membantu siswa lainnya dengan memberikan jawaban di bidang yang Anda pahami. Berbagi dalam forum diskusi ini dapat pula membantu meningkatkan retensi ilmu Anda.

Peraturan paling penting adalah bersikap sopan dan memperlakukan semua siswa lain dan instruktur, dengan hormat. Pelanggaran etika ini dapat berakibat pada dikeluarkannya Anda dari kelas.

Anda akan memiliki akses ke forum diskusi kelas selama Anda aktif terdaftar ke kelas ini. Saat masa belajar Anda sudah habis, maka forum diskusi tidak akan dapat diakses. Namun, jika Anda sukses menyelesaikan kelas (lulus dan sampai mendapatkan sertifikat kompetensi dari Dicoding), maka Anda tetap dapat mengakses forum diskusi kelas ini walau masa belajar Anda untuk kelas ini telah habis.

### Sebelum Mulai Bertanya

1. Sebelum membuat diskusi baru, gunakan fitur pencarian untuk cek diskusi lama terlebih dahulu. Problem Anda mungkin pernah dibahas dan dijawab solusinya, dalam forum. Dengan begitu, Anda pun tidak perlu menghabiskan waktu untuk bertanya dan untuk menunggu jawaban kembali.
2. Namun jika tidak menemukan diskusi lama dengan topik yang sama dengan masalah Anda, silakan membuat diskusi baru. Simak cara dan langkah berikut ini: <https://help.dicoding.com/academy-dicoding/aturan-membuat-pertanyaan-di-forum-diskusi-kelas/>

### Fitur Bermanfaat

* **Mencari Posting**  
  Anda dapat mencari topik atau diskusi menggunakan tiga cara yang tersedia pada halaman utama forum diskusi.
  1. Gunakan fitur “Cari Judul Diskusi” untuk mencari topik berdasarkan judul atau pertanyaannya.
  2. Gunakan fitur “Pilih Berdasarkan Modul” untuk melihat diskusi berdasarkan modul kelasnya.
  3. Gunakan fitur “Keyword Populer” untuk melihat diskusi berdasarkan Keyword yang paling sering ditanyakan.
* **Upvote dan Downvote Komentar pada Diskusi**  
  Saat membuka sebuah diskusi, Anda akan menemukan komentar beserta jawaban dari siswa dan instruktur di bagian bawah. Pada setiap komentar terdapat tombol panah atas (˄) dan panah bawah (˅). Klik tombol panah atas (˄) jika Anda merasa komentar atau jawaban tersebut bijaksana, menarik, dan bermanfaat. Gunakan tombol panah bawah (˅) jika Anda merasa komentar atau jawaban tersebut tidak sopan, tidak membantu, atau bahkan tidak tepat.  
    
  Gunakan Fitur Upvote dan Downvote ini seperti arahan di atas, sehingga kualitas komentar dan jawaban pada forum diskusi dapat terjaga dengan baik.
* **Approved Answer**  
  Jika terdapat komentar/tanggapan berupa solusi terbaik dari permasalahan yang Anda tanyakan, maka tandai komentar tersebut sebagai jawaban terpilih. Caranya dengan menekan tombol “Jawaban Terpilih” di sisi kanan atas komentar.  
    
  Fitur ini juga akan mempermudah siswa lainnya dengan pertanyaan yang sama untuk menemukan titik terang solusinya.

### Mengenal Teman Sekelas Anda

Berhubungan baik dengan teman sekelas dan instruktur merupakan bagian penting dari kelas ini karena mereka dapat membantu jika Anda mengalami kendala dalam pemahaman materi.

Oleh karena itu, kami ingin Anda meluangkan waktu untuk mencairkan suasana dan saling mengenal. Membangun interaksi dengan siswa lain akan membuat pengalaman belajar Anda jauh lebih menyenangkan dan menarik.

Beberapa hal yang dapat Anda tulis pada perkenalan diri:

* Siapa Anda dan dari mana Anda berasal?
* Apa pekerjaan atau pendidikan Anda saat ini?
* Kenapa Anda mengambil pelatihan ini? Apakah mungkin karena Anda sedang mengejar perubahan dalam karir, atau lainnya?

|  |
| --- |
| Ayo aktif di forum diskusi kelas ini!  Yuk mulai dengan memperkenalkan diri Anda pada thread Perkenalan Diri berikut:  <https://www.dicoding.com/academies/256/discussions/88855> |

## Glosarium

### C

**Compiler**

Program komputer yang menerjemahkan kode yang ditulis dalam satu bahasa pemrograman ke bahasa lain yang dimengerti oleh mesin.

### D

**Debugging**

Proses mengidentifikasi dan menghilangkan error pada aplikasi.

### I

**Interpreter**

Program untuk menerjemahkan setiap baris kode dari bahasa pemrograman menjadi bahasa yang dimengerti oleh mesin secara langsung. Tanpa melalui proses kompilasi.

### L

**Library**

Sekumpulan sumber daya, biasanya berupa kode atau program, yang memiliki fungsionalitas tertentu dan bisa digunakan pada program lain.

### R

**Runtime**

Lingkungan, platform, atau sumber daya yang menjalankan suatu kode/program.

### S

**String Concatenation**

Proses penggabungan dua String atau lebih menggunakan operator penjumlahan (+).

**String Interpolation**

Proses mengganti placeholder di dalam String dengan nilai dari variabel tertentu.

## Daftar Referensi

[1]  Indeed. “Gaji di Indonesia - gaji rata-rata” id.indeed.com. <https://id.indeed.com/career/front-end-developer/salaries?from=top_sb> (diakses pada 2 Juni 2022)

[2]  Indeed. “Gaji di Indonesia - gaji rata-rata” id.indeed.com. <https://id.indeed.com/career/back-end-developer/salaries?from=top_sb> (diakses pada 2 Juni 2022)

[3] D. Flanagan, "The JavaScript Standard Library," dalam JavaScript: The Definitive Guide, 7th Edition. O'Reilly Media, Inc., 2020. Available: O'Reilly

[4] D. Flanagan, "Introduction to JavaScript," dalam JavaScript: The Definitive Guide, 7th Edition. O'Reilly Media, Inc., 2020. Available: O'Reilly

[5] S. Amarasinghe. "Reading 17: Concurrency". web.mit.edu. <http://web.mit.edu/6.005/www/fa14/classes/17-concurrency/> (diakses pada 22 Maret 2021)

[6] M. Lee. "What's the difference between a tilde (~) and a caret (^) in a npm package.json file?". michaelsoolee.com. <https://michaelsoolee.com/npm-package-tilde-caret/> (diakses pada 26 Maret 2021)

# Pengenalan JavaScript

Sebelum mulai belajar suatu hal yang baru, penting bagi Anda untuk memahami apa yang akan dipelajari dan apa yang bisa dilakukan dengannya. Dalam hal ini kita akan mulai mengenal JavaScript.

Pada akhir modul ini, Anda diharapkan dapat:

* Mengetahui apa itu JavaScript
* Mengetahui sejarah singkat dan perkembangan JavaScript
* Mengerti beberapa alasan untuk mulai mempelajari JavaScript

## Apa itu JavaScript

JavaScript adalah bahasa pemrograman tingkat tinggi yang pada awalnya dikembangkan untuk membuat website menjadi lebih “hidup”. Bersama dengan HTML dan CSS, JavaScript menjadi bahasa pemrograman paling populer untuk mengembangkan aplikasi berbasis web. Bahasa ini mampu memberikan logic ke dalam website, sehingga website tersebut memiliki fungsionalitas tambahan dan lebih interaktif.

Awalnya JavaScript dibuat supaya dapat berjalan di lingkungan browser dan membuat website menjadi lebih interaktif. Namun, saat ini Anda sebagai developer dapat menggunakan bahasa pemrograman JavaScript untuk mengembangkan berbagai macam platform. Sehingga, tidak hanya sebatas browser/client, tetapi JavaScript juga bisa berjalan di luar browser menggunakan Node.js.

JavaScript termasuk ke dalam kategori scripting language. Apa maksudnya? Salah satu ciri-ciri utama dari bahasa scripting adalah kode tidak perlu dikompilasi agar bisa dijalankan. Scripting language menggunakan interpreter untuk menerjemahkan kode atau perintah yang kita tulis supaya dimengerti oleh mesin.

Itulah kenapa bahasa scripting tidak membutuhkan banyak kode yang perlu ditulis agar sebuah program bisa dijalankan. Hanya dengan satu baris kode berikut Anda sudah bisa membuat program yang menampilkan teks “Hello, World!” ke layar.

1. console.log(“Hello, World!”);

Sederhana, bukan? Itulah salah satu ciri utama JavaScript sebagai scripting language. Tentu tidak lengkap rasanya ketika kita sudah mengetahui apa itu JavaScript, tetapi tidak tahu sejarah di baliknya. Sejarah singkat dari JavaScript dapat Anda simak di materi berikutnya.

## Sejarah JavaScript

JavaScript dibuat pada tahun 1995 oleh Brendan Eich, programmer dari Netscape. Bahasa ini awalnya dinamai “Mocha” kemudian berubah menjadi “LiveScript”. Pada saat itu bahasa Java merupakan bahasa pemrograman yang paling populer. Untuk memanfaatkan kepopulerannya, nama LiveScript pun diubah menjadi “JavaScript”. Jadi, meskipun namanya mirip, JavaScript tidak ada hubungannya dengan bahasa pemrograman Java.

Setelah diadopsi di luar Netscape, JavaScript distandarisasi oleh European Computer Manufacturers Association (ECMA). Sejak saat itu JavaScript juga dikenal dengan ECMAScript. Meskipun begitu, masih banyak yang menyebutnya dengan JavaScript hingga saat ini.

Terdapat beberapa versi JavaScript yang sudah distandarisasi oleh ECMAScript. Pada tahun 2000 hingga 2010, ECMAScript 3 merupakan versi yang banyak digunakan ketika JavaScript sedang mendominasi. Selama waktu tersebut, ECMAScript 4 sedang dalam proses pengembangan dengan harapan akan memberikan improvisasi yang cukup signifikan. Namun, ambisi tersebut tidak berjalan mulus sehingga pada tahun 2008 pengembangan ECMAScript dihentikan.

Walaupun begitu, ini bukan akhir dari JavaScript. Pengembangan berganti menjadi ECMAScript 5 dengan mengurangi ambisinya dan hanya melakukan perbaikan pada hal yang tidak jadi kontroversi. Pembaruan tersebut berhasil dan akhirnya ECMAScript 5 rilis pada tahun 2009.

Lalu, pada tahun 2015 ECMAScript 6 rilis dengan membawa perubahan yang cukup besar termasuk ide-ide yang sudah direncanakan untuk versi 4. Sejak saat itu, tiap tahun JavaScript melakukan update bersifat minor.

## Kenapa JavaScript

Jadi, kenapa kita perlu mempelajari JavaScript?

Alasan utamanya karena JavaScript merupakan bahasa yang penting untuk Anda kuasai jika ingin menjadi web developer, baik itu front-end maupun back-end.

Berikut ini adalah beberapa kelebihan dari JavaScript yang dapat Anda pertimbangkan sebelum mulai mempelajari JavaScript:

* **JavaScript bahasa yang versatile**  
  JavaScript bisa berjalan di lingkungan browser, server, bahkan desktop. Artinya, jika Anda bisa menguasai bahasa ini, maka skill Anda bisa digunakan di mana pun.
* **Mudah dipelajari oleh pemula**  
  JavaScript termasuk salah satu bahasa pemrograman yang ramah bagi pemula. Anda tidak perlu menginstal software dan lingkungan pengembangan lain yang rumit untuk memulai membuat program dengan JavaScript. Cukup dengan browser Anda sudah bisa menulis kode JavaScript dan menjalankannya sekaligus.  
    
  Selain itu, sebagai salah satu bahasa pemrograman paling populer, JavaScript memiliki komunitas yang besar pada situs seperti [StackOverflow](https://stackoverflow.com/questions/tagged/javascript) yang siap membantu Anda jika memiliki pertanyaan atau kesulitan dalam JavaScript.
* **Potensi karir yang meyakinkan**  
  Mengikuti perkembangan teknologi dan banyaknya bisnis yang mulai merambah ke ranah digital, JavaScript menjadi salah satu skill yang paling banyak dicari di industri. Jika Anda mencari kata kunci “JavaScript” pada laman pencarian kerja seperti JobStreet, akan muncul hampir 1.500 lowongan pekerjaan di Indonesia yang bisa Anda lamar.  
    
  Bukan hanya ramai peminat, pekerjaan yang berkaitan dengan JavaScript juga dihargai cukup tinggi. Menurut data yang diambil dari situs id.indeed.com rata-rata gaji seorang Front End Developer adalah Rp 7.300.000 per bulan [[1](https://id.indeed.com/career/front-end-developer/salaries?from=top_sb)] dan untuk Back End Developer adalah Rp 9.200.000 per bulan [[2](https://id.indeed.com/career/back-end-developer/salaries?from=top_sb)].

Bagaimana menurut Anda? Apakah setelah mengetahui beberapa kelebihan dari JavaScript membuat Anda semakin semangat untuk mulai mempelajarinya? Tunggu apalagi, yuk kita mulai belajar JavaScript!

## Rangkuman Materi

Sebelum mempelajari sesuatu tentunya penting untuk mengetahui apa yang akan kita pelajari serta bagaimana sejarahnya dan bagaimana perkembangannya.

Kita telah membahas beberapa hal di modul ini, antara lain:

* JavaScript dibuat dengan tujuan awal agar website menjadi lebih interaktif.
* JavaScript termasuk ke dalam kategori scripting language, sehingga kode tidak perlu dikompilasi untuk bisa dijalankan. Terdapat interpreter untuk menerjemahkan kode kita agar bisa dimengerti oleh mesin.
* Terdapat dua lingkungan umum untuk menjalankan JavaScript, yaitu browser dan Node.js
* JavaScript dikembangkan dengan standar ECMAScript. Update besar terakhir tersaji dalam versi ES6 pada tahun 2015. Sejak saat itu, tiap tahun JavaScript melakukan update bersifat minor.

## Video Pengenalan JavaScript

Untuk memperdalam dan mempermudah pemahaman pada materi ini, Anda dapat menyimak video pembahasannya berikut.

<https://youtu.be/Uwvl-X8wiYc>

### Materi Pendukung

Ulas beberapa sumber di bawah ini untuk mempelajari lebih lanjut tentang JavaScript:

* [JavaScript MDN Web Docs](https://developer.mozilla.org/en-US/docs/Web/JavaScript)
* [An Introduction to JavaScript](https://javascript.info/intro)
* [Introduction to Node.js](https://nodejs.dev/en/learn/)
* [Differences between Node.js and the Browser](https://nodejs.dev/en/learn/differences-between-nodejs-and-the-browser/)

# [Fundamental JavaScript](https://www.dicoding.com/academies/256/tutorials/13863)

## Pengenalan Fundamentals JavaScript

Setelah kita mengetahui apa itu JavaScript dan sejarah singkat di baliknya, pada modul ini Anda akan mulai menuliskan kode JavaScript pertama. Sudah tidak sabar, bukan? Yuk, kita mulai belajar.

Pada akhir modul ini, Anda diharapkan dapat:

* Menuliskan kode JavaScript dan menampilkannya ke konsol
* Memahami penggunaan komentar pada kode
* Memahami konsep variabel pada JavaScript
* Memahami beberapa jenis tipe data pada JavaScript
* Mengenal operator pada JavaScript dan bagaimana menggunakannya
* Memahami konsep percabangan pada pemrograman
* Memahami konsep perulangan pada pemrograman

## Menulis Kode JavaScript Pertama

Menulis kode dalam suatu bahasa pemrograman sederhananya adalah menuliskan instruksi-instruksi untuk dijalankan oleh komputer. Kode di bawah ini merupakan instruksi bagi terminal atau konsol untuk mencatat (log) kalimat “Hello, World!”.

1. console.log("Hello World!");

console.log adalah kode bawaan JavaScript untuk menampilkan pesan ke konsol, bisa berupa web konsol atau konsol dari terminal/command prompt.

[](https://www.dicoding.com/academies/256/tutorials/13863)

Kode atau teks yang berada di dalam tanda kurung adalah pesan yang ingin ditampilkan. Pada contoh kode di atas, kita menggunakan tanda kutip (“”) untuk menandakan bahwa pesan yang ingin ditampilkan merupakan sebuah string atau teks. Kita akan membahas mengenai tipe data pada beberapa modul ke depan. Saat ini kita akan fokus untuk menampilkan teks terlebih dulu. Sekarang cobalah untuk menampilkan nama Anda sendiri ke konsol.



JavaScript membagi instruksi secara umum menjadi dua jenis, yaitu expression dan statement.

Expression adalah unit kode yang dapat dievaluasi menjadi suatu nilai. Untuk lebih mudahnya, perhatikan contoh kode berikut:

1. 5; // Baris kode ini merupakan expression karena interpreter akan membaca kode ini dan menghasilkan nilai 5.
2. 2 + 3; // Baris kode ini juga merupakan expression. Interpreter mengevaluasi kode dan juga akan menghasilkan nilai 5.

Sementara itu, statement adalah instruksi untuk melakukan sebuah aksi tertentu. Aksi ini bisa termasuk membuat variabel, melakukan perulangan, pengecekan kondisi, dll.

1. var yourName;
2. let yourAge;
3. const numberOfDays;  // declaration statement

Jika Anda masih bingung untuk membedakan expression dan statement, tidak perlu khawatir. Saat ini belum banyak kode yang bisa kita jadikan contoh. Namun, konsep ini akan banyak kita gunakan pada materi-materi yang akan datang. Yang perlu Anda ingat adalah:

* Expression merupakan kode yang menghasilkan nilai.
* Statement menunjukkan aksi yang dilakukan.

Melalui materi yang kami sampaikan di atas seharusnya Anda sudah tahu cara menampilkan teks di console. Untuk pemanasan menuju materi lebih dalam selanjutnya, yuk mulai selesaikan tantangan berikut.

Cobalah cetak teks "Saya mulai belajar JavaScript!" pada console melalui teks editor yang kami sediakan di bawah ini.

**Catatan:**  Pastikan pesan yang ditulis indentik dengan memperhatikan penggunaan huruf besar dan kecil.

Berikut beberapa catatan dalam menggunakan editor tersebut,

1. Klik tombol **Jalankan** untuk melihat hasilnya.
2. Klik tombol  **Submit Kode** untuk mengirim jawaban (kode) agar dapat melanjutkan pembelajaran.

****

## Comments

Pada materi sebelumnya kita telah membahas bahwa kode yang kita tulis merupakan kumpulan instruksi yang harus dijalankan oleh komputer. Salah satu instruksi yang penting adalah memberi tahu komputer untuk mengabaikan perintah yang kita tulis. Instruksi yang ditulis dalam suatu program tetapi tidak dijalankan oleh komputer disebut “comments”.

Sebuah komentar akan dilewatkan oleh interpreter atau compiler, sehingga tidak akan memengaruhi alur program yang kita tulis. Komentar ini bisa digunakan sebagai dokumentasi atau penjelasan dari kode yang kita tulis.

Terdapat dua metode untuk memberikan komentar. Pertama, untuk memberikan komentar pada satu baris saja, kita bisa gunakan tanda dua garis miring (//) di awal baris.



Lalu, untuk memberikan komentar lebih dari satu baris kita bisa menggunakan tanda /\* sebagai pembuka komentar dan tanda \*/ untuk penutup komentar. Teks apa pun yang berada di antara tanda tersebut akan dijadikan komentar dan tidak akan dieksekusi.

****

## Variable

Ketika menulis sebuah program, kita memberi tahu komputer cara memproses informasi seperti mencetak teks ke layar atau melakukan operasi perhitungan. Untuk lebih mudah dalam penggunaan dan pemanggilan data, kita bisa memanfaatkan variabel. Variabel umumnya digunakan untuk menyimpan informasi atau nilai yang akan dikelola dalam sebuah program.

Pada JavaScript setidaknya ada tiga cara untuk mendeklarasikan sebuah variabel, yaitu menggunakan keyword var, let, dan const. Pada versi ECMAScript 2015 (ES6) dikenalkan deklarasi variabel dengan let dan const untuk menggantikan var yang dinilai kontroversial dan rawan menimbulkan bug.

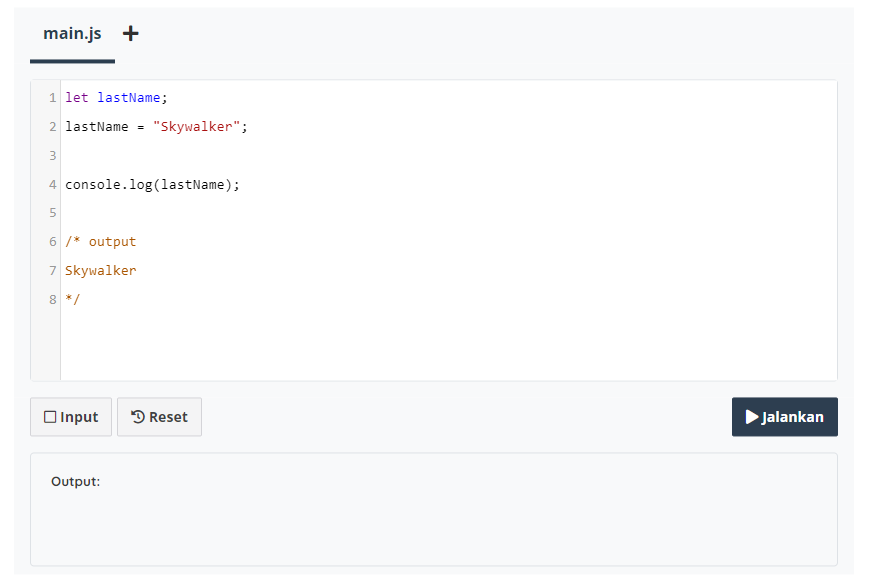
Dalam kelas ini, kita akan banyak menggunakan keyword let dan const. Jika Anda penasaran kenapa var sudah tidak lagi disarankan, silakan simak diskusi [berikut](https://softwareengineering.stackexchange.com/questions/274342/is-there-any-reason-to-use-the-var-keyword-in-es6).

Sekarang bagaimana caranya membuat sebuah variabel dalam JavaScript? Cukup mudah. Ketikkan keyword let yang diikuti dengan nama variabelnya.

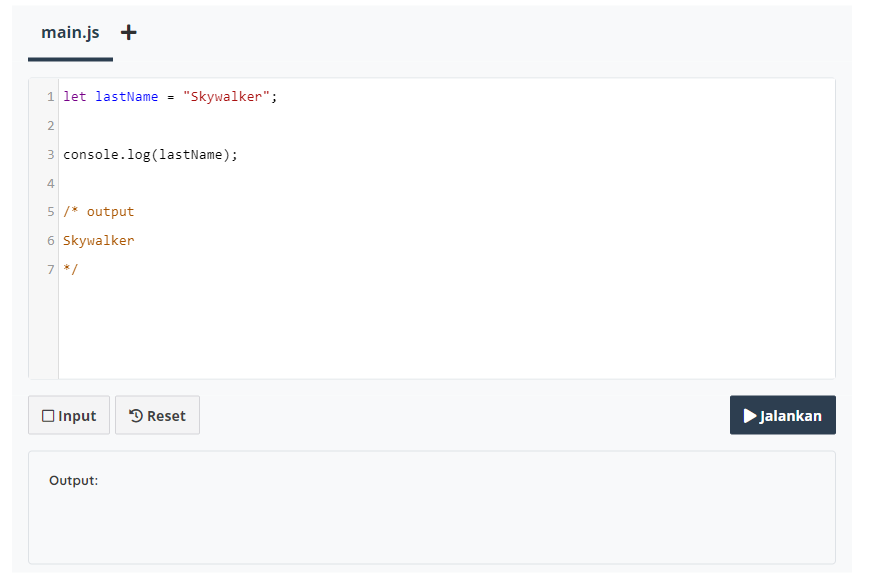
1. let lastName;

Kode untuk mendeklarasikan variabel seperti di atas juga dikenal dengan declaration statement.

Selanjutnya, Anda bisa mengisi nilai variabel di atas menggunakan tanda sama dengan (=).



Anda juga bisa langsung menginisialisasi nilai variabel setelah mendeklarasikannya seperti berikut:

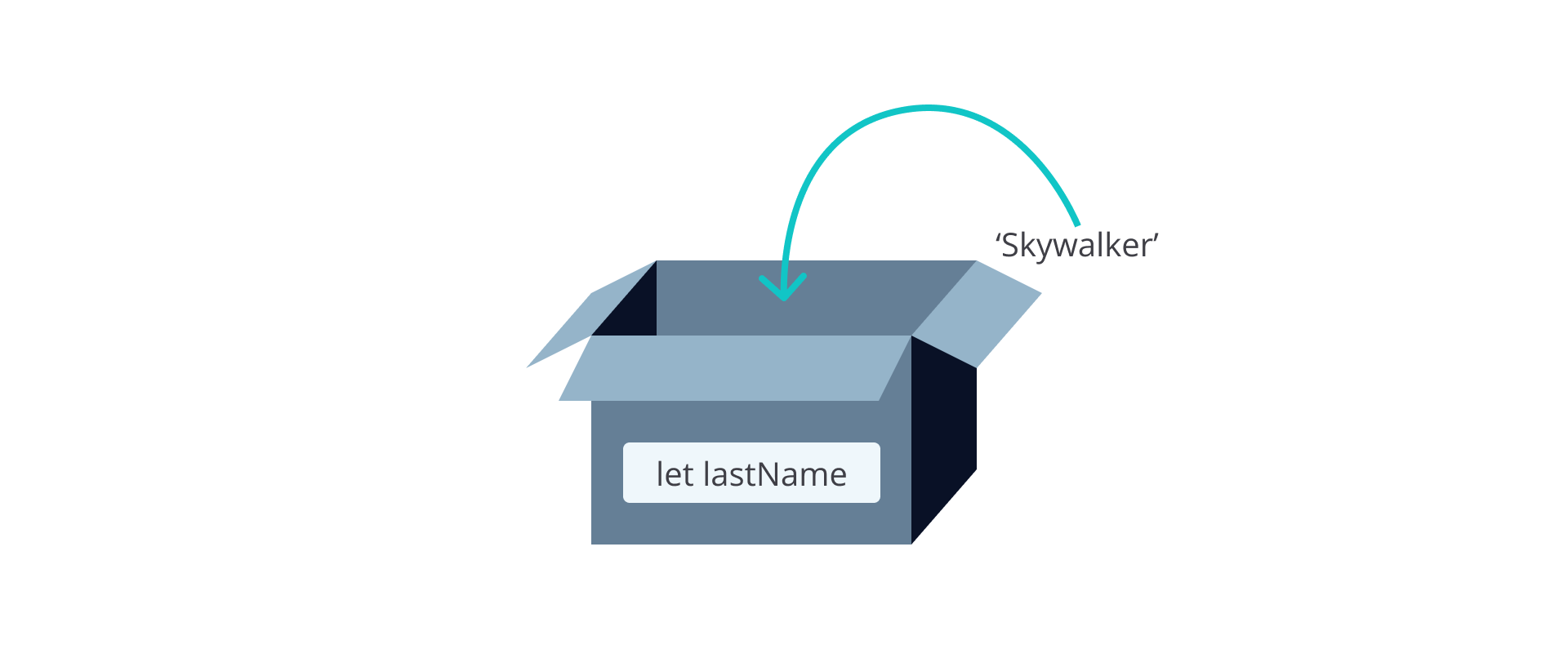


Kode untuk menginisialisasikan nilai ke dalam sebuah variabel dengan tanda sama dengan (=) ini disebut dengan assignment expression.

Tunggu sebentar. Kita kembali bertemu dengan istilah statement dan expression. Karena expression pasti menghasilkan nilai, sehingga mereka bisa muncul di mana pun dalam program JavaScript. Sementara itu, statement merujuk pada aksi. Sehingga, pada bagian kode tertentu yang membutuhkan nilai tidak bisa kita isi dengan sebuah statement. Contohnya seperti kode berikut:

1. let fullName = let lastName; // Error karena let lastName adalah sebuah statement untuk deklarasi variabel. Statement tidak bisa berada di posisi expression.
2. let fullName = (lastName = "Skywalker"); // (lastName = "Skywalker") merupakan expression, sehingga kode ini tidak error.
3. let fullName = "Luke" + "Skywalker"; // "Luke" + "Skywalker" juga merupakan expression, sehingga kode ini tidak error.

Melalui contoh kode di atas, kita bisa bayangkan variabel sebagai sebuah kotak atau wadah yang menyimpan nilai. Proses inisialisasi atau assignment berarti kita memasukkan nilai ke dalam kotak tersebut.

[](https://www.dicoding.com/academies/256/tutorials/13872?from=13867)

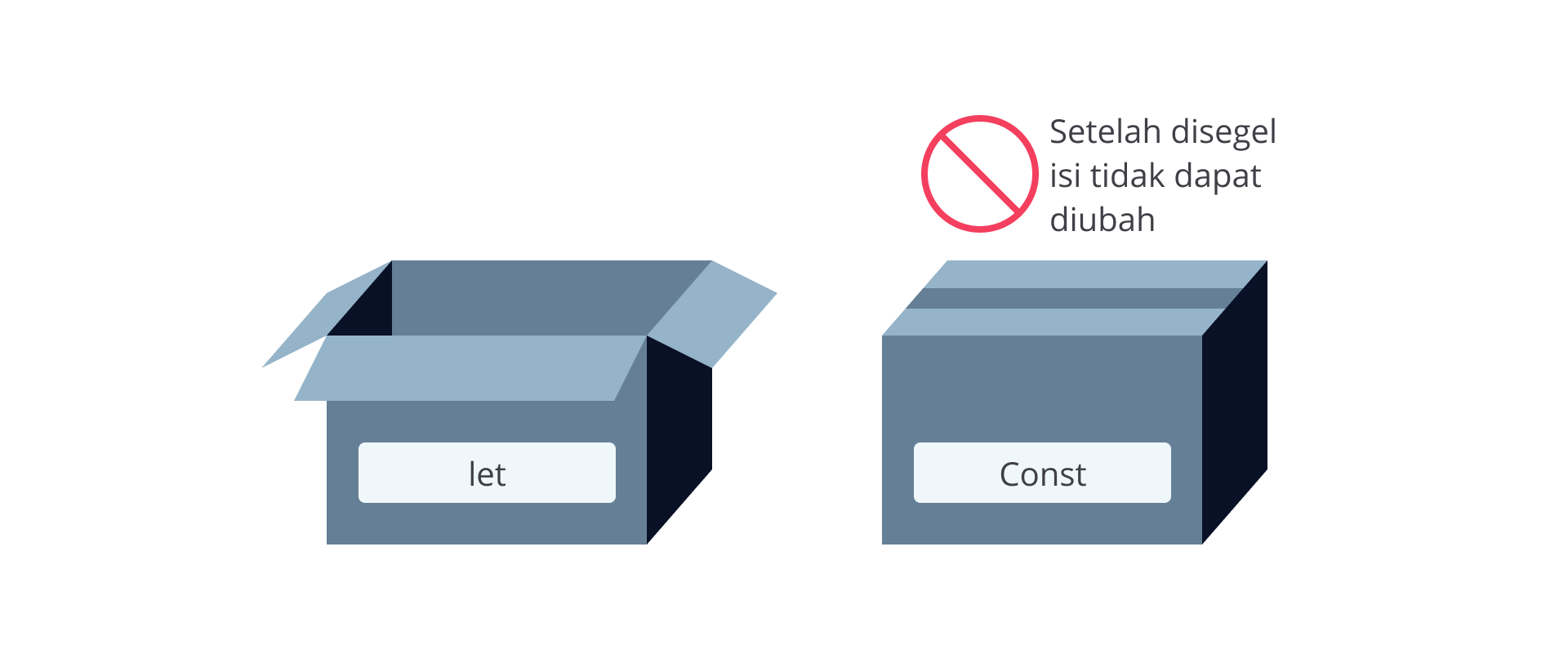
Variabel lastName di atas akan tersimpan di dalam memori komputer.

Setiap variabel memiliki nama yang dapat kita panggil dan gunakan. Kita dapat menamai variabel dengan nama apa pun, tetapi pastikan penamaannya masih masuk akal dengan konteksnya supaya kode mudah di-maintenance.

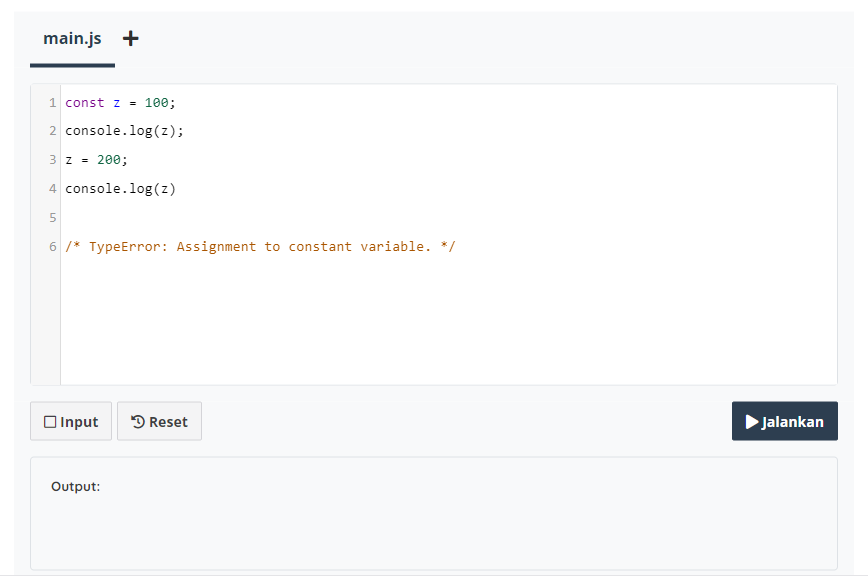
Sebaiknya hindari penamaan variabel dengan istilah umum seperti “data”. Gunakanlah penamaan variabel yang dapat mendeskripsikan nilai dari variabel itu sendiri. Berikut beberapa aturan dalam penamaan variabel yang perlu Anda ketahui:

* Harus dimulai dengan huruf atau underscore (\_).
* Dapat terdiri dari huruf, angka, dan underscore (\_) dengan berbagai kombinasi.
* Tidak boleh mengandung spasi (whitespace). Jika penamaan variabel lebih dari dua kata, tuliskan secara camelCase. Contoh firstName, lastName, catName, dll.
* Tidak boleh mengandung karakter spesial (! . , / \ + \* = dll.)

Lalu, bagaimana dengan const? Const merupakan kependekan dari constant. Artinya, kita akan mendeklarasikan sebuah variabel dengan const ketika ingin variabel bernilai konstan dan tidak bisa diubah setelah diinisialisasi nilainya. Bayangkan variabel bernilai const sebagai sebuah kotak yang ditutup dan disegel setelah diisi, sehingga nilainya tidak bisa diubah lagi.

[](https://www.dicoding.com/academies/256/tutorials/13872?from=13867)

Jika menginisialisasi kembali nilai variabel yang menggunakan const, kita akan mendapati eror “TypeError: Assignment to constant variable.”

****

## Tipe Data

Pada materi sebelumnya kita telah belajar tentang variabel untuk menyimpan nilai. Nilai yang kita tetapkan pada variabel pasti memiliki tipe data. Tipe data merupakan pengklasifikasian data berdasarkan jenisnya. Pada JavaScript terdapat beberapa tipe data sebagai berikut:

### Undefined

Tipe data ini terbentuk ketika sebuah variabel tidak memiliki nilai. Artinya, ketika kita mendeklarasikan variabel tanpa menginisialisasikan nilainya, variabel tersebut menjadi undefined. Contoh:



Pada contoh kode di atas, kita mendeklarasikan variabel x, tetapi kita tidak menginisialisasikan dengan nilai apa pun. Ketika kita memastikan tipe data dengan menggunakan fungsi typeof(), ia akan menghasilkan output undefined.

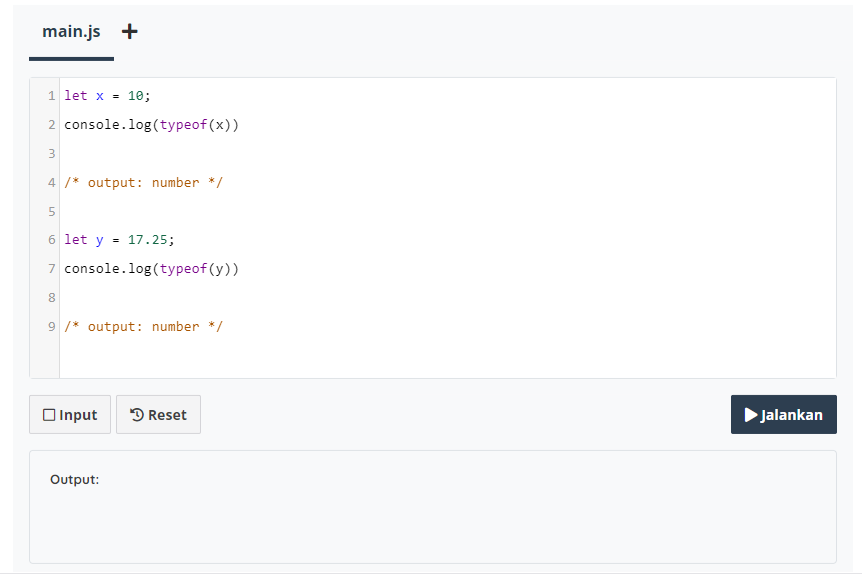
Fungsi typeof() digunakan untuk memastikan tipe data pada variabel dengan mengembalikan tipe data tersebut dalam bentuk teks.

### Numbers

Nilai dari tipe data number adalah angka. Variabel bertipe data number dituliskan seperti angka pada umumnya:

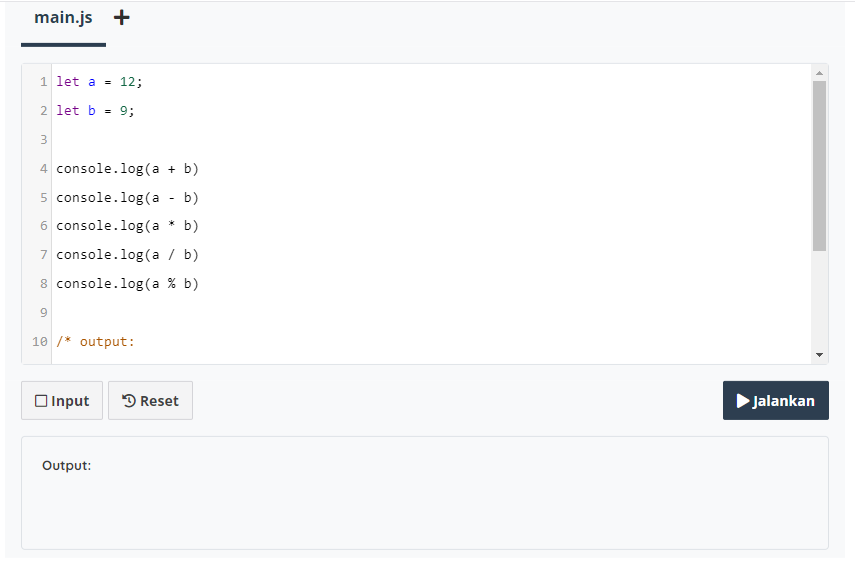
1. let x = 10;

Jika angka tersebut merupakan sebuah bilangan desimal, maka kita bisa gunakan tanda titik pada pecahan bilangannya.



Pada tipe data number, kita juga dapat melakukan perhitungan aritmatika seperti penjumlahan, pengurangan, perkalian, dsb. Berikut operator yang dapat kita gunakan dalam perhitungan aritmatika pada tipe data number:

| **Operator** | **Fungsi** | **Contoh** |
| --- | --- | --- |
| + | Penjumlahan | 10 + 10 = 20 |
| - | Pengurangan | 15 - 7 = 8 |
| / | Pembagian | 21 / 7 = 3 |
| \* | Perkalian | 9 \* 9 = 81 |
| % | Sisa hasil bagi | 5 % 2 = 1 |
| \*\* | Perpangkatan | 3 \*\* 3 = 27 |



Pada operator aritmatika juga terdapat operator increment (++) dan decrement (--). Operator increment dan decrement digunakan untuk menambahkan atau mengurangi nilai 1 pada nilai variabel yang ada sekarang.

Operator ini dapat dituliskan sebelum atau sesudah variabel, tetapi hal tersebut bukan berarti sama. Berikut ketentuannya:

* Jika dituliskan setelah variabel (x++), expression akan menghasilkan nilai variabel sebelum ditingkatkan nilainya.
* Jika dituliskan sebelum variabel (++x), expression akan menghasilkan nilai variabel setelah ditingkatkan nilainya.

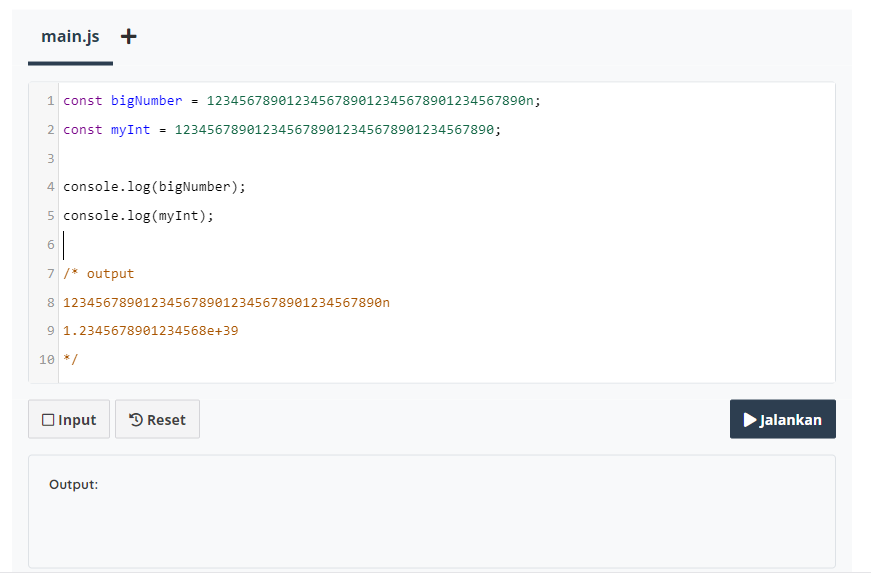
Untuk lebih jelasnya, berikut adalah contoh kode penerapan operator tersebut, perhatikan hasil yang didapat.



### BigInt

Pada JavaScript, tipe data “Number” hanya mencakup nilai dari -(253 - 1) hingga (253 - 1). Untuk kebutuhan umum, sebenarnya nilai tersebut sudah sangat cukup. Namun, akan ada kebutuhan tertentu di mana kita membutuhkan cakupan nilai yang lebih besar, seperti untuk kriptografi atau menentukan waktu hingga presisi microsecond.

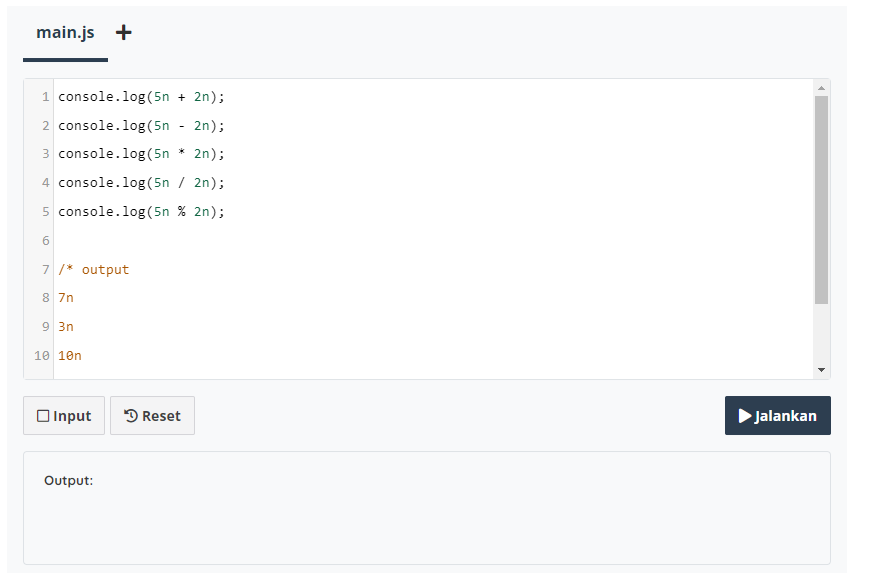
Untuk nilai di luar Number, kita bisa menggunakan tipe BigInt. Untuk membedakan tipe BigInt dan Number, tambahkan karakter n di akhir angka. Contohnya adalah seperti kode di bawah ini. Bandingkan dengan yang bertipe Number.



Meskipun digunakan untuk menyimpan angka dengan nilai besar, namun BigInt tetap bisa digunakan untuk nilai yang lebih kecil.

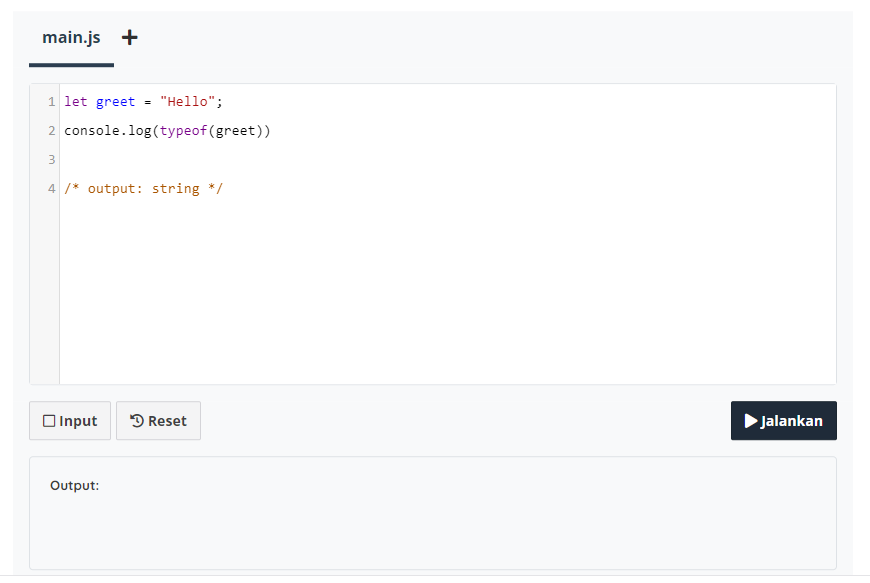
1. const bigIntButSmall = 7n;

Kita juga bisa menggunakan BigInt untuk operasi aritmatika pada umumnya. Yang membedakan adalah pada operasi pembagian, hasilnya akan dibulatkan ke bawah dan tanpa mengandung nilai desimal. Contohnya adalah seperti ini:



### **Strings**

Tipe data selanjutnya adalah string yang merupakan sebuah teks. Untuk menetapkan nilai sebagai string pada variabel gunakan tanda petik satu (‘) atau petik dua (“) di antara teksnya. Contohnya:



Tidak ada perbedaan antara menggunakan petik satu atau petik dua. Anda dapat menggunakan tanda petik secara bergantian, khususnya jika Anda memiliki teks yang mengandung tanda petik.



Lalu bagaimana jika teks memiliki kedua tanda petik seperti ini?



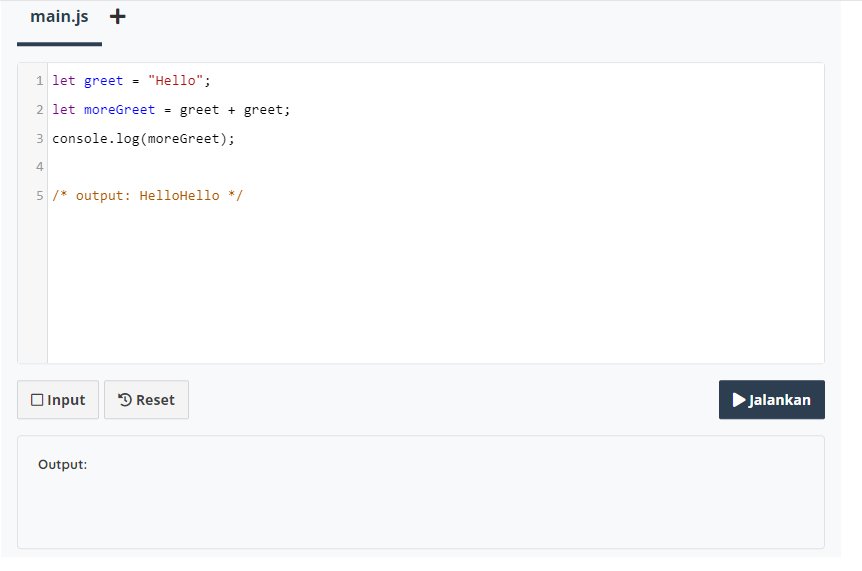
Tentunya kode di atas akan menghasilkan eror. Solusinya, gunakan backslash(\) untuk mengurangi ambiguitas dalam tanda petik. Mekanisme ini juga dikenal dengan nama escape string. Sehingga kode di atas akan menjadi seperti berikut:

1. const answer = '"I think it\'s awesome!" he answered confidently';

Backslash sebelum tanda petik akan memberitahukan interpreter bahwa itu hanyalah tanda petik dan tidak boleh ditafsirkan sebagai pembatas string. Selain tanda petik, backslash juga berguna untuk mengabaikan simbol lain yang menimbulkan ambigu di dalam string, contohnya seperti backslash itu sendiri.

1. console.log("Windows path: C:\\Program Files\\MyProject");

Pada String, kita juga dapat menggunakan operator plus (+). Operator tersebut berfungsi untuk menggabungkan dua teks yang terpisah menjadi satu buah teks. Contohnya seperti ini:



Ingat, string concatenation seperti di atas akan menggabungkan string apa adanya, sehingga jika Anda ingin menggabungkan dua kata atau lebih perlu menambahkan spasi sendiri.

Selain concatenation, string pada JavaScript juga mendukung string interpolation. Sederhananya, kita bisa memasukkan variabel ke dalam sebuah string template. Contohnya adalah seperti berikut:



Perhatikan bahwa untuk mendefinisikan string template, Anda perlu menggunakan backticks (`), biasanya terletak di keyboard di bawah tombol Esc . Di dalam string letakkan variabel yang ingin dimasukkan ke dalam placeholder ${myName}.

### **Boolean**

Boolean hanya memiliki dua nilai, yaitu **true** atau **false**. Tipe data ini menjadi kunci utama dalam penentuan logika. Kita akan banyak menggunakannya nanti dalam materi if/else statement. Untuk menetapkan nilai boolean pada variabel, gunakan keyword true atau false seperti di bawah ini.



Kita juga bisa menggunakan operator komparasi seperti lebih dari (>) atau kurang dari (<). Contohnya:



### **Null**

Tipe berikutnya adalah null. Serupa dengan undefined, namun null perlu diinisialisasikan pada variabel. Null biasa digunakan sebagai nilai sementara pada variabel, tapi sebenarnya nilai tersebut “tidak ada”.

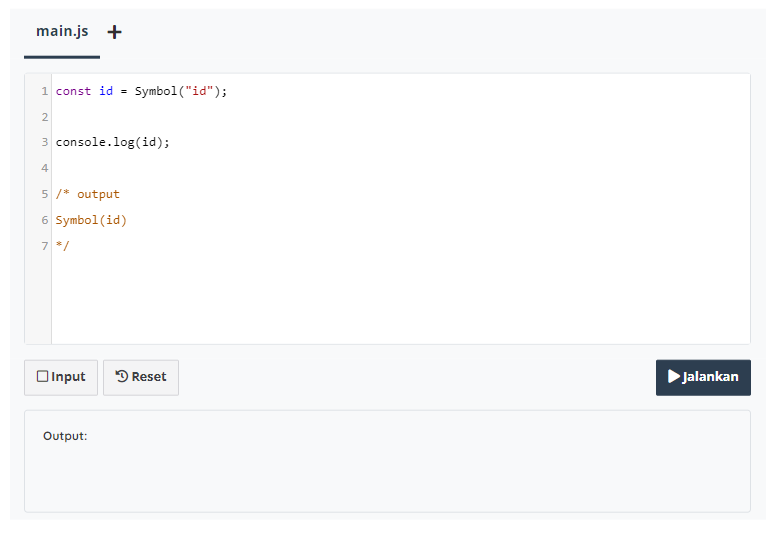
Terkadang kita perlu membuat sebuah variabel, namun kita belum memerlukan nilai apa-apa dan tidak ingin terikat oleh tipe data apa pun. Nah, daripada kita tidak menetapkan nilai apa pun (variabel akan undefined) sebaiknya kita beri nilai null pada variabel tersebut dan ubah nanti ketika kita membutuhkannya.

Untuk menetapkan null pada variabel, kita dapat gunakan keyword null ketika variabel tersebut diinisialisasi.



### **Symbol**

Symbol adalah tipe data baru yang dikenalkan pada ES6. Tipe data Symbol digunakan untuk menunjukkan identifier yang unik. Ketika membuat Symbol, kita bisa memberikan deskripsi atau nama symbol seperti ini:



Symbol disebut sebagai identifier yang unik karena meskipun kita membuat dua variabel symbol dengan nama atau deskripsi yang sama, kedua nilainya tetap dianggap berbeda. Contohnya lihat kode berikut:



Symbol ini umumnya digunakan sebagai nama property dari Object. Object sendiri merupakan tipe data kompleks untuk menyimpan berbagai struktur data. Kita akan segera bertemu dan mempelajari tentang object pada modul Data Structure.

## Operator

Pada materi ini kita akan mempelajari tentang operator yang terdapat pada JavaScript. Operator dalam bahasa pemrograman sendiri adalah simbol yang memberi tahu interpreter untuk melakukan operasi seperti matematika, relasional, atau logika untuk memberikan hasil tertentu.

### **Assignment Operator**

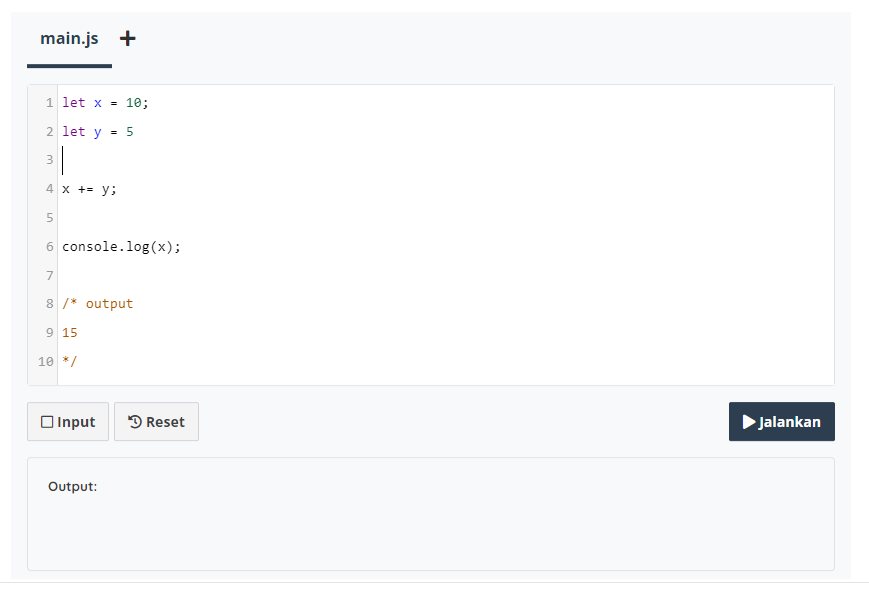
Dari contoh kode yang kita gunakan sebelumnya, sebenarnya kita sudah menggunakan assignment operator. Operator ini digunakan untuk memberikan nilai pada variabel.

Pada dasarnya operator ini adalah tanda sama dengan (=), di mana tanda ini digunakan untuk menginisialisasi nilai pada variabel. Tempatkan variabel yang ingin diberi nilai di sebelah kiri, sementara nilainya di sebelah kanan. Di antara keduanya terdapat operator assignment.

1. x = y;

Expression di atas berarti kita menginisialisasikan nilai y pada variabel x, sehingga nilai x sekarang memiliki nilai yang sama dengan y.

Ada beberapa assignment operator tambahan lain dalam menginisialisasikan nilai pada variabel. Kita bisa menyebutnya sebagai shortcut dalam menentukan nilai. Contohnya:



Pada contoh kode di atas, terdapat expression x += y; Apa artinya? Assignment operator tersebut digunakan sebagai shortcut dari x = x + y. Cara ini juga dapat digunakan pada operator aritmatika lain seperti, perkalian, pengurangan, pembagian, dan lainnya.

1. let x = 10;
2. let y = 5;
4. x += y; // artinya -> x = x + y;
5. x -= y; // artinya -> x = x - y;
6. x \*= y; // artinya -> x = x \* y;
7. x /= y; // artinya -> x = x / y;
8. x %= y; // artinya -> x = x % y;

### Comparison Operator

Sekarang kita sudah mengetahui bagaimana cara menyimpan nilai pada sebuah variabel. Nah, selanjutnya kita akan belajar mengenai operator komparasi sebagai logika dasar dalam membandingkan nilai pada JavaScript.

Terdapat serangkaian karakter khusus yang disebut dengan operator pembanding/komparasi yang dapat mengevaluasi dan membandingkan dua nilai. Berikut daftar operator dan fungsinya:

| **Operator** | **Fungsi** |
| --- | --- |
| == | Membandingkan kedua nilai apakah sama (tidak identik). |
| != | Membandingkan kedua nilai apakah tidak sama (tidak identik). |
| === | Membandingkan kedua nilai apakah identik. |
| !== | Membandingkan kedua nilai apakah tidak identik. |
| > | Membandingkan dua nilai apakah nilai pertama lebih dari nilai kedua. |
| >= | Membandingkan dua nilai apakah nilai pertama lebih atau sama dengan nilai kedua. |
| < | Membandingkan dua nilai apakah nilai pertama kurang dari nilai kedua. |
| <= | Membandingkan dua nilai apakah nilai pertama kurang atau sama dengan nilai kedua. |

Ketika kita melakukan perbandingan antara dua nilai, JavaScript akan mengevaluasi kedua nilai tersebut dan mengembalikan boolean dengan nilai hasil perbandingan tersebut, baik false atau true. Berikut contohnya:



#### Perbedaan antara “Sama” dan “Identik”

Dalam operator komparasi di JavaScript, hal yang menjadi sedikit “tricky” adalah membedakan antara “sama” (==) dan “identik” (===).

Kita sudah mengetahui bahwa setiap nilai pasti memiliki tipe data baik itu number, string atau boolean. Contohnya sebuah string “10” dan number 10 merupakan hal yang serupa, tetapi keduanya tidak benar-benar sama.

Hal inilah yang membedakan antara sama dan identik pada JavaScript. Jika kita ingin membandingkan hanya dari kesamaan nilainya kita bisa gunakan == tapi jika kita ingin membandingkan dengan memperhatikan tipe datanya kita gunakan ===.

Contohnya seperti berikut:



### Logical Operator

Terdapat beberapa operator lain yang dapat kita gunakan untuk menetapkan logika yang lebih kompleks, yakni dengan logical operators. Dengan logical operator, kita dapat menggunakan kombinasi dari dua nilai boolean atau bahkan lebih dalam menetapkan logika.

Pada JavaScript terdapat tiga buah karakter khusus yang berfungsi sebagai logical operator. Berikut macam-macam logical operator dan fungsinya:

| **Operator** | **Deskripsi** |
| --- | --- |
| && | Operator dan (and). Logika akan menghasilkan nilai true apabila semua kondisi terpenuhi (bernilai true). |
| || | Operator atau (or). Logika akan menghasilkan nilai true apabila ada salah satu kondisi terpenuhi (bernilai true). |
| ! | Operator tidak (not). Digunakan untuk membalikkan suatu kondisi. |

Berikut contoh penerapannya pada JavaScript:



Mungkin sebagian dari kita bertanya, sebenarnya apa kegunaan dari nilai boolean selain hanya menampilkan nilai true dan false saja? Pada pembahasan tipe data sudah pernah disebutkan bahwa boolean merupakan salah satu kunci dari logika pemrograman, karena boolean dapat mengontrol bagaimana alur program kita akan berjalan.

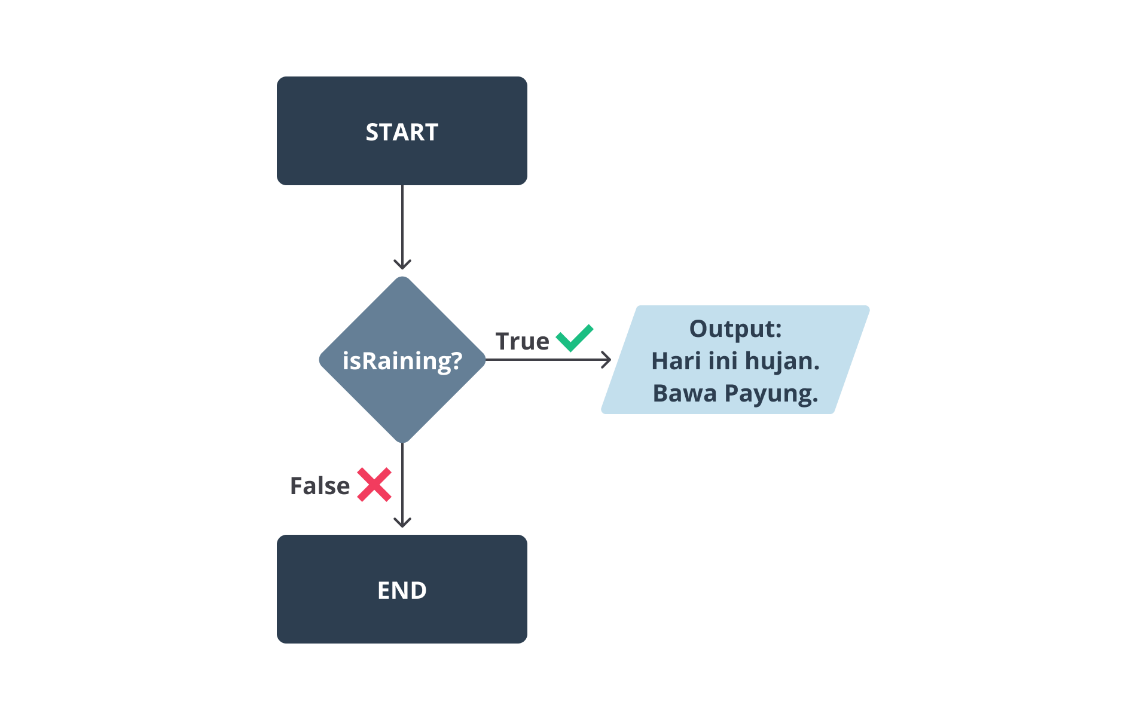
Lantas bagaimana cara boolean mengontrol sebuah aliran program? Pada materi selanjutnya, kita akan membahas mengenai if/else statement yang dapat mengontrol flow pada program, tentunya pada penggunaan statement boolean ini sangat berguna.

## If/Else Statement

Setiap hari kita melakukan perhitungan dan perbandingan guna membuat keputusan, apa pun itu. Contohnya, apakah perlu mencuci kendaraan ketika cuaca sedang cerah? Apa saja transportasi online yang bisa dipesan ketika hujan untuk sampai di tempat tujuan?

Ketika mengembangkan sebuah program, kita akan bertemu dengan alur bercabang tergantung pada kondisi yang terjadi. Untuk mengakomodasi dan mengecek sebuah kondisi dalam JavaScript, kita menggunakan kata kunci **if**.

Statement if akan menguji suatu kondisi. Jika kondisi bernilai true, maka blok kode di dalamnya akan dijalankan. Sebaliknya, jika bernilai false, maka proses yang ditentukan akan dilewatkan.

[](https://www.dicoding.com/academies/256/tutorials/13883?from=13882)

Flowchart di atas jika diterjemahkan menjadi kode, akan menjadi seperti berikut:



const isRaining = true;

console.log("Persiapan sebelum berangkat kegiatan.");

if (isRaining) {

console.log("Hari ini hujan. Bawa payung.");

}

console.log("Berangkat kegiatan.");

/\* output:

Persiapan sebelum berangkat kegiatan.

Hari ini hujan. Bawa payung.

Berangkat kegiatan.

\*/

Jika Anda mengubah nilai isRaining menjadi false, maka kode di dalam blok kode if akan dilewatkan. Sehingga program Anda tidak akan mengingatkan untuk membawa payung.

Lalu bagaimana jika Anda ingin melakukan operasi lain ketika kondisi bernilai false? Jawabannya adalah statement else. Pada contoh kode berikut kita akan melakukan pengecekan kondisi menggunakan operator perbandingan.



Terdapat variabel x dengan nilai 50, kemudian kita bertanya, “Hai JavaScript! Apakah x lebih dari 70?” Jika kondisi tersebut benar, maka kita dapat memerintahkan JavaScript untuk menampilkan nilainya. Jika salah, kita perintahkan JavaScript untuk menampilkan teks “Nilai kurang dari 70”.

Kita juga bisa mengecek beberapa kondisi sekaligus dengan menggabungkan else dan if. Contohnya adalah seperti program berikut:



Pengecekan kondisi akan dilakukan dari statement if paling awal. Sehingga, ketika nilai language adalah “French”, maka pengecekan untuk language === “Japanese” tidak akan dilakukan.

Selain if statement di atas, JavaScript juga mendukung ternary operator atau conditional expressions. Dengan ini, kita bisa menuliskan if-else statement hanya dalam satu baris.



Ternary operator membutuhkan tiga operand. Sebelum tanda tanya (?) berisi kondisi yang ingin kita evaluasi. Kemudian diikuti dengan expression apabila nilai kondisinya benar sebelum tanda titik dua. Terakhir adalah expression yang dieksekusi ketika kondisinya salah. Karena merupakan conditional expression, maka operand kedua dan ketiga harus mengembalikan nilai.

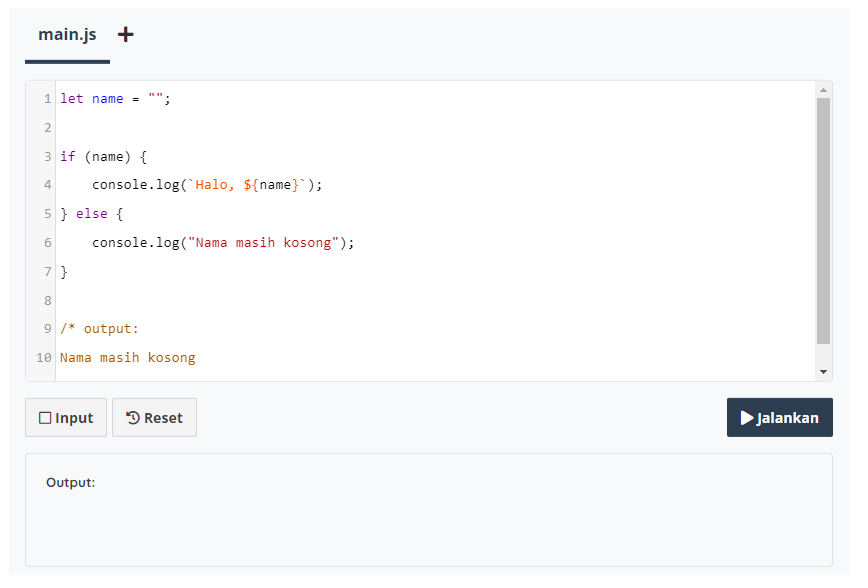
### **Truthy & Falsy**

Di dalam if statement kita perlu memasukkan expression yang akan dievaluasi. Umumnya, expression tersebut mengembalikan nilai boolean untuk menentukan kondisi true atau false. Lalu bagaimana jika kita menuliskan expression yang tidak mengembalikan nilai boolean? Jawabannya bisa.

Setiap nilai pada JavaScript pada dasarnya juga mewarisi sifat boolean. Nilai ini dikenal dengan truthy atau falsy. Nilai truthy berarti nilai yang ketika dievaluasi akan menghasilkan nilai true, begitu pula falsy bernilai false. Jadi manakah yang termasuk truthy dan falsy? Selain nilai boolean false, tipe data atau nilai yang dianggap falsy, antara lain:

* Number 0
* BigInt 0n
* String kosong seperti “” atau ‘’
* null
* undefined
* NaN, atau Not a Number

Selain contoh di atas maka nilainya adalah truthy dan ketika dievaluasi ke dalam if statement akan bernilai true. Berikut ini contohnya dalam kode:

****

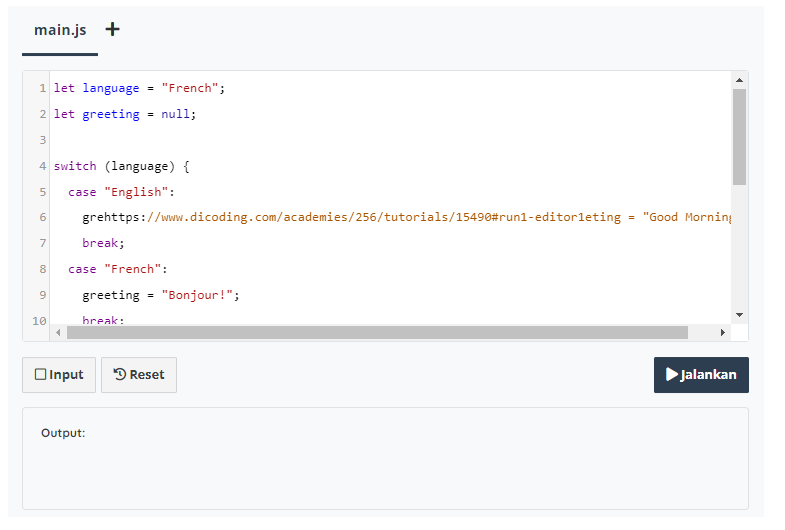
## Switch Case Statement

Sebelumnya kita telah mempelajari bagaimana percabangan logika menggunakan if statement. Selain if, JavaScript juga mendukung switch statement untuk melakukan pengecekan banyak kondisi dengan lebih mudah dan ringkas.

1. switch (expression) {
2. case value1:
3. // do something
4. break;
5. case value2:
6. // do something
7. break;
8. ...
9. ...
10. default:
11. // do something else
12. }

Tanda kurung setelah keyword switch berisi variabel atau expression yang akan dievaluasi. Kemudian untuk setiap kondisi yang mungkin terjadi, kita masukkan keyword case diikuti dengan nilai yang valid. Jika kondisi pada case sama dengan variabel pada switch, maka blok kode setelah titik dua (:) akan dijalankan. Keyword break digunakan untuk keluar dari proses switch. Terdapat satu case bernama default yang memiliki fungsi yang sama dengan keyword else pada control flow if-else. Jika tidak ada nilai yang sama dengan variabel pada switch, maka blok kode ini akan dijalankan.

Berikut ini adalah contoh kode dari materi if-else yang dikonversi menjadi statement switch:



let language = "French";

let greeting = null;

switch (language) {

case "English":

grehttps://www.dicoding.com/academies/256/tutorials/15490#run1-editor1eting = "Good Morning!";

break;

case "French":

greeting = "Bonjour!";

break;

case "Japanese":

greeting = "Ohayou Gozaimasu!";

break;

default:

greeting = "Selamat Pagi!";

}

console.log(greeting);

/\* output

Bonjour!

\*/

1

## Loop

Ketika menulis program komputer, akan ada situasi di mana kita perlu melakukan hal yang sama berkali-kali. Misalnya kita ingin menampilkan semua nama pengguna yang terdaftar di aplikasi atau sesederhana menampilkan angka 1 sampai 10. Tentunya tidak praktis jika kita menulis kode seperti berikut:

1. console.log(1);
2. console.log(2);
3. console.log(3);
4. console.log(4);
5. console.log(5);
6. console.log(6);
7. console.log(7);
8. console.log(8);
9. console.log(9);
10. console.log(10);

Bagaimana jika kita perlu menampilkan angka 1 sampai 100? Sudah terbayang repotnya, bukan? Maka dari itu kita perlu mempelajari teknik yang dapat mengatasi permasalahan tersebut, teknik ini disebut dengan looping. JavaScript memiliki banyak opsi untuk melakukan looping atau perulangan kode, antara lain:

### **For loop**

Dari beberapa cara melakukan proses loop pada JavaScript, “for” merupakan salah satu cara yang banyak digunakan. Struktur dasar dari for tampak seperti berikut:

1. for(inisialisasi variabel; test kondisi; perubahan nilai variabel) {
2. // do something
3. }

Berikut ini contoh penerapannya secara nyata:



Lebih ringkas, bukan? Mungkin kode tersebut sulit dipahami, jadi mari kita bahas sedikit demi sedikit. Terdapat tiga bagian utama dalam sintaks for di atas:

* Pertama, variabel i sebagai index iterasi. Pada variabel ini kita menginisialisasi nilai awal dari perulangan.
* Kedua, operasi perbandingan. Pada bagian ini, JavaScript akan melakukan pengecekan kondisi apakah perulangan masih perlu dilakukan. Jika bernilai true, maka kode di dalam blok for akan dijalankan.
* Ketiga, increment/decrement. Di sini kita melakukan penambahan atau pengurangan variabel iterasi. Jadi, pada contoh di atas variabel i akan ditambah dengan 1 di setiap akhir perulangan. Perubahan nilai ini penting karena jika kita mengubah nilainya, proses perulangan dapat terus berjalan selama kondisinya terpenuhi.

JIka diartikan, maka kode di atas bisa dimaknai dengan “Jika i kurang dari 5, maka tampilkan nilai i.”

### **For of loop**

Cara lain dalam melakukan looping adalah menggunakan for..of. For of mulai hadir pada ECMAScript 2015 (ES6). Cara ini jauh lebih sederhana dan modern dibanding for loop biasa. Sintaks dasar dari for of loop adalah seperti ini:

1. for(arrayItem of myArray) {
2. // do something
3. }

Yup, for of tidak membutuhkan banyak statement untuk melakukan looping pada array. Penjelasan tentang array akan lebih detail dibahas pada modul berikutnya. Sebagai permulaan, kita bisa menganggap array sebagai kumpulan nilai yang disimpan dalam satu variabel.

Dengan for..of nilai tiap array akan diinisialisasi pada variabel baru yang kita tentukan pada tiap proses looping-nya. Jumlah proses looping-nya pun akan menyesuaikan dengan ukuran dari array. Sederhananya seperti kita melakukan perintah “Hei JavaScript! Lakukan perulangan pada myArray, akses tiap nilainya, dan simpan pada variabel arrayItem”. Pada proses looping kita gunakan variabel arrayItem untuk mengakses tiap nilai dari item myArray.

Agak sulit memang menjelaskan dengan kata-kata, mari kita terjemahkan dalam kode secara langsung.



Lebih mudah dan simpel bukan?

### **While and do-while**

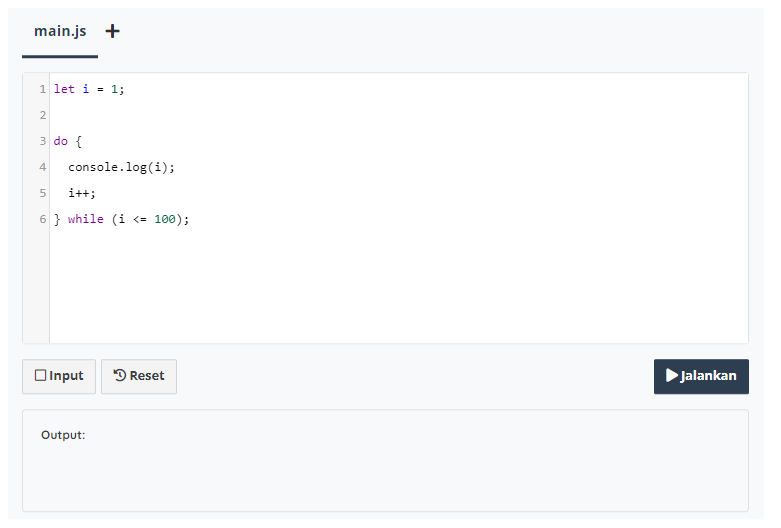
Metode lain untuk melakukan looping adalah dengan statement while. Sama seperti for, instruksi while mengevaluasi ekspresi boolean dan menjalankan kode di dalam blok while ketika bernilai true.

Untuk menampilkan angka 1 sampai 100 dengan while kita bisa menulis kode seperti berikut:



Bisa dilihat pada kode di atas bahwa looping dengan while tidak memiliki ketergantungan dengan variabel iterasi seperti pada for loop. Karena itu, meskipun while dapat melakukan perulangan yang sama dengan for, while lebih cocok digunakan pada kasus di mana kita tidak tahu pasti berapa banyak perulangan yang diperlukan.

Bentuk lain dari while adalah perulangan do-while.



Kondisi pada while akan dievaluasi sebelum blok kode di dalamnya dijalankan, sedangkan do-while akan mengevaluasi boolean expression setelah blok kodenya berjalan. Ini artinya kode di dalam do-while akan dijalankan setidaknya satu kali.

### **Infinite loops**

Ketika menerapkan perulangan pada program, ada satu kondisi yang perlu kita hindari yaitu infinite loop. Infinite loop atau endless loop adalah kondisi di mana program kita stucked di dalam perulangan. Ia akan berjalan terus hingga menyebabkan crash pada aplikasi dan komputer kecuali ada intervensi secara eksternal, seperti mematikan aplikasi.

Kode berikut ini adalah contoh di mana kondisi infinite loop dapat terjadi:



* [**While**](https://www.dicoding.com/academies/256/tutorials/13887?from=15490#tab1-code1)
* [For](https://www.dicoding.com/academies/256/tutorials/13887?from=15490#tab1-code2)

1. let i = 1;
3. while (i <= 5) {
4. console.log(i);
5. }



* [While](https://www.dicoding.com/academies/256/tutorials/13887#tab1-code1)
* [**For**](https://www.dicoding.com/academies/256/tutorials/13887#tab1-code2)

1. for(let i = 1; i <= 5; i=1) {
2. console.log(i);
3. }

Dapatkah Anda menemukan apa yang salah dari kode di atas sehingga terjadi infinite loop?

Jawabannya adalah karena variabel i selalu bernilai 1. Alhasil, kondisi i <= 5 akan selalu bernilai true yang mengakibatkan aplikasi akan terus mencetak 1 ke konsol sehingga mengalami crash.

## Kuis Coding : Variabel dan Tipe Data

Untuk menguji kemampuan praktikal Anda dalam memahami materi variabel dan tipe data di JavaScript, silakan selesaikan kuis berikut.

Sebelum Anda mengerjakan kuis, ada beberapa hal yang perlu Anda perhatikan.

* Mohon untuk membaca secara seksama perintah, kriteria, atau soal pada komentar kode yang diberi tanda **TODO.**
* Lakukan pengujian pada kode yang Anda tulis contohnya dengan menggunakan console.log() untuk memastikan hasil sesuai dengan yang Anda harapkan. Gunakan tombol **Jalankan**.

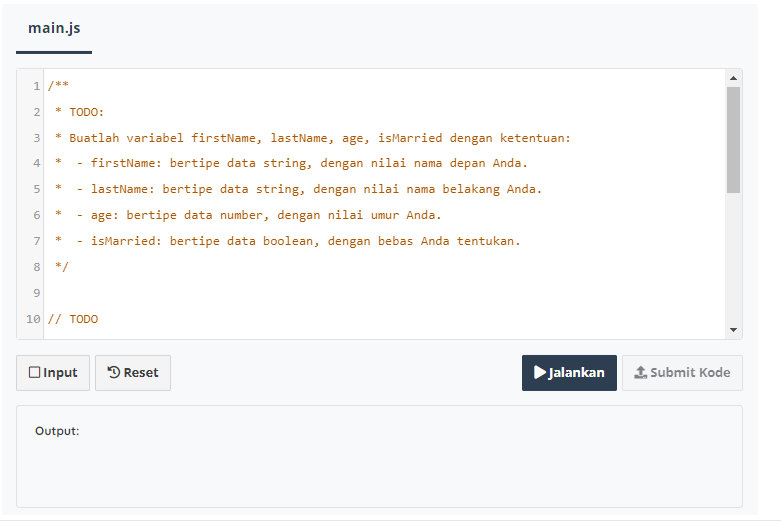
Bila sudah yakin dengan pekerjaan yang Anda lakukan, silakan submit kode dengan klik tombol **Submit kode**. Jika kode yang di-submit benar, Anda bisa melanjutkan ke modul berikutnya. Jika tidak, Anda bisa memperbaiki kembali kode sebanyak yang Anda mau.

Batas waktu tunggu untuk mengirimkan kembali perbaikan kode adalah 1 menit. Jadi pastikan kode yang Anda tulis sudah memenuhi perintah, kriteria, atau soal pada kuis kali ini.

Good luck!

|  |
| --- |
| **Mengalami kendala?** Cobalah kunjungi [**Forum Diskusi**](https://www.dicoding.com/academies/256/discussions?query=&query_criteria=&sort=&sort_direction=&title=&tutorial=15865&keywords=&creator=&page=1) untuk mencari solusi atau bertanya langsung kepada mentor kami. |

Selamat, Anda telah menyelesaikan modul ini.  
Kode yang Anda kirimkan telah berfungsi dengan baik.



1

/\*\*

2

\* TODO:

3

\* Buatlah variabel firstName, lastName, age, isMarried dengan ketentuan:

4

\* - firstName: bertipe data string, dengan nilai nama depan Anda.

5

\* - lastName: bertipe data string, dengan nilai nama belakang Anda.

6

\* - age: bertipe data number, dengan nilai umur Anda.

7

\* - isMarried: bertipe data boolean, dengan bebas Anda tentukan.

8

\*/

9

​

10

// TODO

11

let firstName = 'Saepul';

12

let lastName = 'Hidayat';

13

let age = 22;

14

let isMarried = false;

15

​

16

/\*\*

17

\* Jangan hapus kode di bawah ini

18

\*/

19

​

20

module.exports = {

21

firstName, lastName, age, isMarried,

22

};

23

​

 Input  Reset

 Jalankan  Submit Kode

## Kuis Coding : Logika Operator dan If Else

Untuk menguji kemampuan praktikal Anda dalam memahami materi Logika Operator dan If Else di JavaScript, silakan selesaikan kuis berikut.

Sebelum Anda mengerjakan kuis, ada beberapa hal yang perlu Anda perhatikan.

* Mohon untuk membaca secara seksama perintah, kriteria, atau soal pada komentar kode yang diberi tanda **TODO.**
* Lakukan pengujian pada kode yang Anda tulis contohnya dengan menggunakan console.log() untuk memastikan hasil sesuai dengan yang Anda harapkan. Gunakan tombol **Jalankan**.

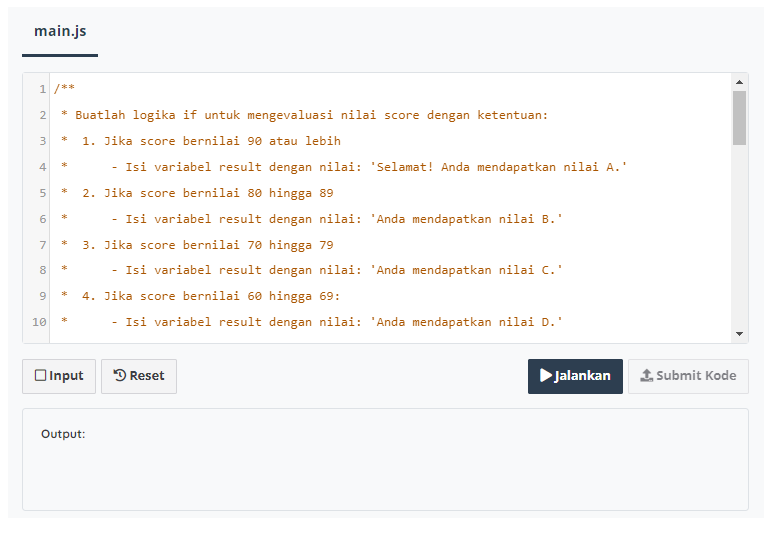
Bila sudah yakin dengan pekerjaan yang Anda lakukan, silakan submit kode dengan klik tombol **Submit kode**. Jika kode yang di-submit benar, Anda bisa melanjutkan ke modul berikutnya. Jika tidak, Anda bisa memperbaiki kembali kode sebanyak yang Anda mau.

Batas waktu tunggu untuk mengirimkan kembali perbaikan kode adalah 1 menit. Jadi pastikan kode yang Anda tulis sudah memenuhi perintah, kriteria, atau soal pada kuis kali ini.

Good luck!

|  |
| --- |
| **Mengalami kendala?** Cobalah kunjungi [Forum Diskusi](https://www.dicoding.com/academies/256/discussions?tutorial=15875) untuk mencari solusi atau bertanya langsung kepada mentor kami. |

Selamat, Anda telah menyelesaikan modul ini.  
Kode yang Anda kirimkan telah berfungsi dengan baik.



1

/\*\*

2

\* Buatlah logika if untuk mengevaluasi nilai score dengan ketentuan:

3

\* 1. Jika score bernilai 90 atau lebih

4

\* - Isi variabel result dengan nilai: 'Selamat! Anda mendapatkan nilai A.'

5

\* 2. Jika score bernilai 80 hingga 89

6

\* - Isi variabel result dengan nilai: 'Anda mendapatkan nilai B.'

7

\* 3. Jika score bernilai 70 hingga 79

8

\* - Isi variabel result dengan nilai: 'Anda mendapatkan nilai C.'

9

\* 4. Jika score bernilai 60 hingga 69:

10

\* - Isi variabel result dengan nilai: 'Anda mendapatkan nilai D.'

11

\* 5. Jika score bernilai di bawah 60:

12

\* - Isi variabel result dengan nilai: 'Anda mendapatkan nilai E.'

13

\*

14

\*

15

\* Note: - Mohon untuk tidak menghapus kode yang sudah ada sebelumnya.

16

\* - Anda tidak perlu membuat variabel result dan score secara manual.

17

\* Gunakan variabel yang sudah disediakan.

18

\*

19

\*/

20

​

## Rangkuman Materi

Pada modul ini kita telah mempelajari logika dan sintaksis dasar JavaScript sebelum membuat aplikasi yang lebih kompleks ke depannya.

Beberapa hal yang telah kita bahas pada modul ini, antara lain:

* Gunakan comments untuk memberitahu interpreter supaya mengabaikan kode atau teks yang kita tulis. Ini akan berguna untuk membuat dokumentasi atau penjelasan atas kode yang kita tulis.
* Kita dapat menyimpan suatu nilai ke dalam variabel. ES6 mengenalkan dua cara baru untuk mendefinisikan variabel, yaitu let dan const. Gunakan const untuk menyimpan nilai yang tidak akan berubah setelah diinisialisasi. Gunakan let apabila nilai di dalam variabel bisa berubah atau diinisialisasi ulang.
* Terdapat tujuh (7) tipe data primitif yang mendefinisikan suatu nilai dalam JavaScript. Ketujuh nilai tersebut antara lain: Undefined, Number, BigInt, String, Boolean, Null, dan Symbol.
* JavaScript memiliki beragam operator yang memberi tahu interpreter untuk melakukan operasi matematika, relasional, atau logika untuk memberikan hasil akhir.
* Pengambilan keputusan adalah hal yang penting dalam pemrograman. Kita bisa memanfaatkan if-else statement atau switch-case untuk memilih satu opsi berdasarkan kondisi yang diberikan.
* Pemrograman juga membantu kita untuk melakukan pekerjaan yang berulang. Dengan kode for atau while, kita bisa melakukan perulangan terhadap suatu kode sebanyak ratusan bahkan ribuan kali hanya dengan beberapa baris saja.

## Video Fundamental JavaScript

Untuk memperdalam dan mempermudah pemahaman pada materi ini, Anda dapat menyimak video pembahasan berikut.

<https://youtu.be/X8SbmPNxVL0>

### Materi Pendukung

Berikut ini adalah beberapa materi tambahan yang bisa Anda pelajari terkait modul JavaScript Fundamentals:

* [Grammar and types](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_Types)
* [Expressions and operators](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Expressions_and_Operators)
* [Control flow and error handling](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Control_flow_and_error_handling)
* [Loops and iteration](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration)

Aturan

Knowledge Check : Fundamental JavaScript bertujuan untuk menguji pengetahuan Anda tentang materi yang telah dipelajari di modul ini.

Terdapat 3 pertanyaan yang harus dikerjakan dalam kuis ini. Beberapa ketentuannya sebagai berikut:

* Syarat nilai kelulusan : 100%
* Durasi ujian : 5 menit

Apabila tidak memenuhi syarat kelulusan, Anda bisa mengulang pengerjaan kuis kembali.

Selamat Mengerjakan!

# Struktur Data

## Pengenalan Struktur Data

Sebelumnya kita sempat mempelajari tentang variabel dan tipe data yang ada di JavaScript. Tipe data yang kita bahas sebelumnya termasuk dalam kategori tipe data primitif yang hanya bisa menyimpan satu nilai.

Pada modul ini kita akan membahas tentang tipe data JavaScript yang dapat menyimpan struktur data yang lebih kompleks. Beberapa jenis struktur data yang akan kita pelajari, antara lain:

* Object
* Array
* Map
* Set

Sudah siap? Mari kita lanjut ke materi selanjutnya!

## Object

Kali ini kita akan berkenalan dengan tipe data object. Sebuah tipe data yang sangat berguna dalam pengembangan aplikasi dengan JavaScript. Object mampu menyimpan nilai dari beragam tipe data dan membentuk data yang lebih kompleks.

Untuk menetapkan objek pada sebuah variabel kita gunakan tanda kurung kurawal {}.

1. const user = {};

Object berisi pasangan key dan value yang juga dikenal dengan property. Key berperan mirip seperti nama variabel yang menyimpan sebuah nilai. Sementara, value berisi nilai dengan tipe data apa pun termasuk objek lain. Key dan value di dalam object dituliskan seperti berikut:

1. let object = {key1: "value1", key2: "value2", key3: "value3"}

Key harus berupa string dan dituliskan sebelum titik dua (:), lalu diikuti dengan value-nya. Meskipun key merupakan string, kita tidak perlu menuliskan tanda petik kecuali ada karakter khusus seperti spasi.

Tanda koma pada properti terakhir bersifat opsional. Namun, jika tanda koma tersebut ditulis akan lebih memudahkan ketika kita ingin memindah, mengubah, atau menghapus properti.

Satu object dapat memiliki beberapa pasang key-value yang dipisahkan dengan tanda koma (**,**).

1. const user = {firstName: "Luke", lastName: "Skywalker", age: 19, isJedi: true};

Dalam menuliskan objek, baris baru tidaklah penting dan tidak akan berpengaruh apa pun. Sehingga lebih baik setiap kita menetapkan key-value buatlah baris baru untuk memisahkan antar nilainya. Hal ini akan memudahkan kita dalam membaca dan memahami struktur data dari sebuah object.

1. const user = {
2. firstName: "Luke",
3. lastName: "Skywalker",
4. age: 19,
5. isJedi: true,
6. };

Kemudian untuk mengakses nilai dari properti object, kita dapat memanggil nama object lalu tanda titik dan diikuti nama propertinya. Contoh:



1

const user = {

2

firstName: "Luke",

3

lastName: "Skywalker",

4

age: 19,

5

isJedi: true,

6

};

7

​

8

console.log(`Halo, nama saya ${user.firstName} ${user.lastName}`);

9

console.log(`Umur saya ${user.age} tahun`);

10

​

11

/\* output

12

Halo, nama saya Luke Skywalker

13

Umur saya 19 tahun

14

\*/

Selain dot operator, kita juga bisa mengakses properti dari object menggunakan bracket atau tanda kurung siku.

1. user[“home world”];

Untuk mengakses key yang memiliki spasi atau karakter khusus lainnya maka kita perlu menggunakan bracket seperti di atas.



1

const user = {

2

firstName: "Luke",

3

lastName: "Skywalker",

4

age: 19,

5

isJedi: true,

6

"home world": "Tattooine"

7

};

8

​

9

console.log(`Halo, nama saya ${user.firstName} ${user.lastName}`);

10

console.log(`Umur saya ${user.age} tahun`);

11

console.log(`Saya berasal dari ${user["home world"]}`);

12

​

13

/\* output

14

Halo, nama saya Luke Skywalker

15

Umur saya 19 tahun

16

Saya berasal dari Tattooine

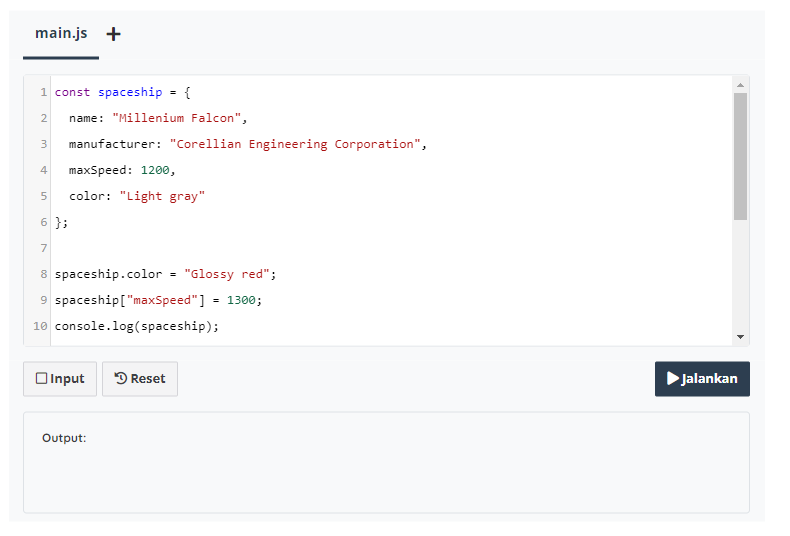
17

\*/

 Input  Reset

 Jalankan

Setelah mempelajari bagaimana membuat object dan menampilkan property di dalamnya, selanjutnya kita akan memodifikasi sebuah object. Untuk mengubah nilai properti di dalam object kita gunakan assignment operator (=).



1

const spaceship = {

2

name: "Millenium Falcon",

3

manufacturer: "Corellian Engineering Corporation",

4

maxSpeed: 1200,

5

color: "Light gray"

6

};

7

​

8

spaceship.color = "Glossy red";

9

spaceship["maxSpeed"] = 1300;

10

console.log(spaceship);

11

​

12

/\* output

13

{

14

name: 'Millenium Falcon',

15

manufacturer: 'Corellian Engineering Corporation',

16

maxSpeed: 1300,

17

color: 'Glossy red'

18

}

19

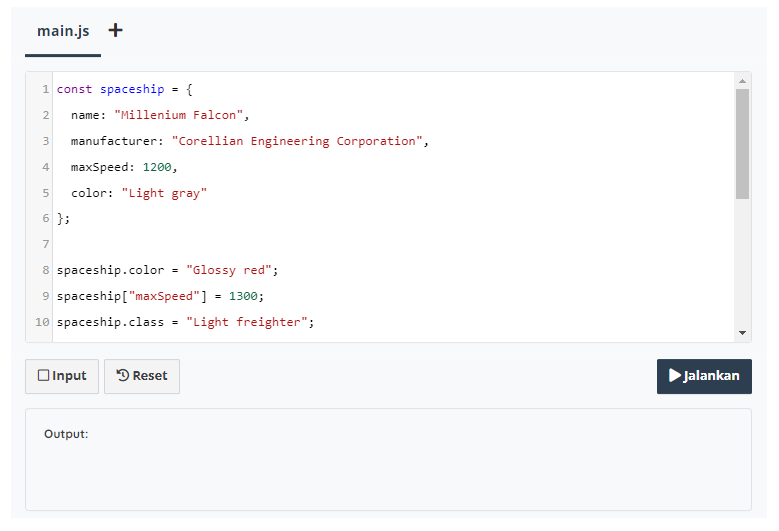
\*/

Tunggu dulu. Object spaceship dideklarasikan sebagai const, tetapi kenapa kita bisa mengubah nilainya?

Yang perlu diperhatikan adalah mengubah nilai berbeda dengan menginisialisasi ulang nilai. Ketika membuat sebuah object, kita tidak terikat dengan properti di dalamnya sehingga kita masih bisa memodifikasi nilainya. Berbeda jika kita menginisialisasi ulang variabel dari object.

1. const spaceship = {
2. name: "Millenium Falcon",
3. manufacturer: "Corellian Engineering Corporation",
4. maxSpeed: 1200,
5. color: "Light gray"
6. };
8. spaceship = { name: "New Millenium Falcon" }; // Error

Ketika kita mengubah object menggunakan assignment operator dan property/key-nya sudah ada, maka nilai di dalamnya akan tergantikan dengan nilai yang baru. Sedangkan, jika property dengan nama key yang ditentukan tidak ditemukan, maka property baru akan ditambahkan ke object.



1

const spaceship = {

2

name: "Millenium Falcon",

3

manufacturer: "Corellian Engineering Corporation",

4

maxSpeed: 1200,

5

color: "Light gray"

6

};

7

​

8

spaceship.color = "Glossy red";

9

spaceship["maxSpeed"] = 1300;

10

spaceship.class = "Light freighter";

11

​

12

console.log(spaceship);

13

​

14

/\* output

15

{

16

name: 'Millenium Falcon',

17

manufacturer: 'Corellian Engineering Corporation',

18

maxSpeed: 1300,

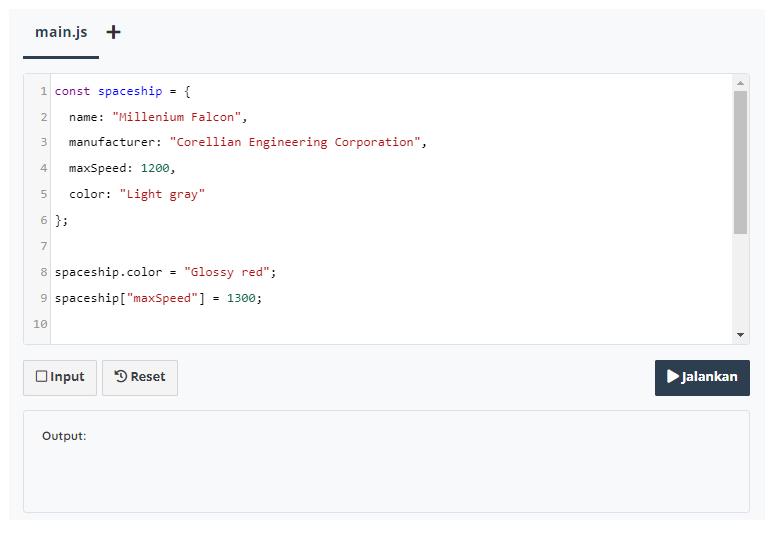
19

color: 'Glossy red',

20

class: 'Light freighter'

Kita juga dapat menghapus property pada object menggunakan keyword delete seperti berikut:



1

const spaceship = {

2

name: "Millenium Falcon",

3

manufacturer: "Corellian Engineering Corporation",

4

maxSpeed: 1200,

5

color: "Light gray"

6

};

7

​

8

spaceship.color = "Glossy red";

9

spaceship["maxSpeed"] = 1300;

10

​

11

delete spaceship.manufacturer;

12

​

13

console.log(spaceship);

14

​

15

/\* output

16

{ name: 'Millenium Falcon', maxSpeed: 1300, color: 'Glossy red' }

17

\*/

## Array

Array merupakan tipe data yang dapat mengelompokkan lebih dari satu nilai dan menempatkannya dalam satu variabel. Contoh:



1

let myArray = ["Cokelat", 42.5, 22, true, "Programming"];

2

console.log(myArray);

3

​

4

/\* output:

5

[ 'Cokelat', 42.5, 22, true, 'Programming' ]

6

\*/

 Input  Reset

 Jalankan

Perbedaan array dengan object adalah data pada array disusun secara berurutan dan diakses menggunakan index. Untuk mengakses nilai di dalam array, kita gunakan tanda kurung siku [] yang di dalamnya berisi angka yang merupakan posisi nilai yang ingin diakses.

1. console.log(myArray[1]);

Jika Anda mengira bahwa konsol akan menampilkan teks ‘Cokelat’, maka tebakan Anda kurang tepat. Dalam sebuah array, indeks dimulai dari 0, sehingga ketika kita mengakses data pada myArray yang berada pada indeks ke-1 artinya data tersebut merupakan data pada posisi ke-2. Jadi nilai yang akan ditampilkan pada konsol adalah 42.5.



1

let myArray = ["Cokelat", 42.5, 22, true, "Programming"];

2

console.log(myArray[1]);

3

​

4

/\* output:

5

42.5

6

\*/

 Input  Reset

 Jalankan

Lalu, apa yang akan terjadi jika kita berusaha mengakses index di luar ukuran array-nya? Jika kita mengakses nilai array lebih dari index-nya, maka hasilnya akan undefined. Index terakhir array selalu jumlah nilai array - 1.



1

let myArray = ["Cokelat", 42.5, 22, true, "Programming"];

2

console.log(myArray[0]);

3

console.log(myArray[1]);

4

console.log(myArray[2]);

5

console.log(myArray[3]);

6

console.log(myArray[4]);

7

console.log(myArray[5]);

8

console.log("Panjang nilai myArray adalah " + myArray.length + ".");

9

​

10

/\* output:

11

Cokelat

12

42.5

13

22

14

true

15

Programming

16

undefined

17

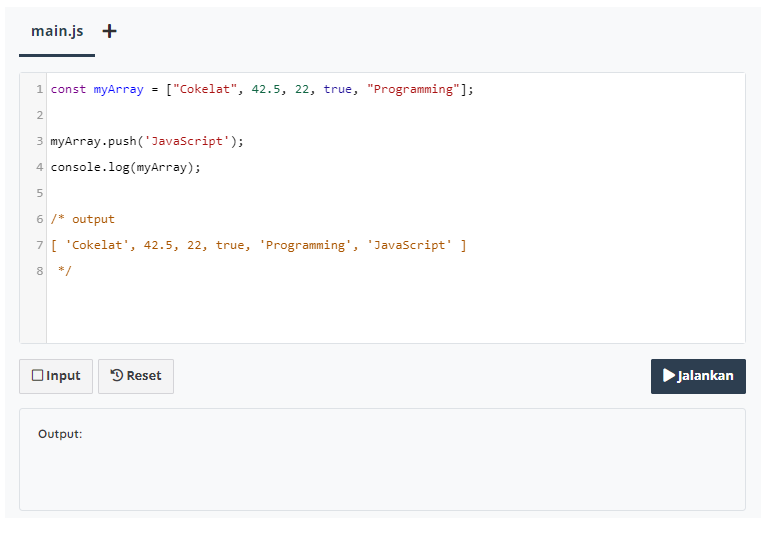
Panjang nilai myArray adalah 5.

18

\*/

Sejauh ini kita baru belajar menginisialisasi dan mengakses elemen dari sebuah array. Pastinya Anda bertanya, “Bagaimana kita memanipulasi data pada array tersebut?”

Nah, untuk menambahkan data ke dalam array, kita bisa menggunakan metode push(). Fungsi push ini akan menambahkan data di akhir array.



1

const myArray = ["Cokelat", 42.5, 22, true, "Programming"];

2

​

3

myArray.push('JavaScript');

4

console.log(myArray);

5

​

6

/\* output

7

[ 'Cokelat', 42.5, 22, true, 'Programming', 'JavaScript' ]

8

\*/

 Input  Reset

 Jalankan

Sedangkan untuk mengeluarkan data atau elemen terakhir dari array, kita bisa gunakan metode pop().



1

const myArray = ["Cokelat", 42.5, 22, true, "Programming"];

2

​

3

myArray.pop();

4

console.log(myArray);

5

​

6

/\* output

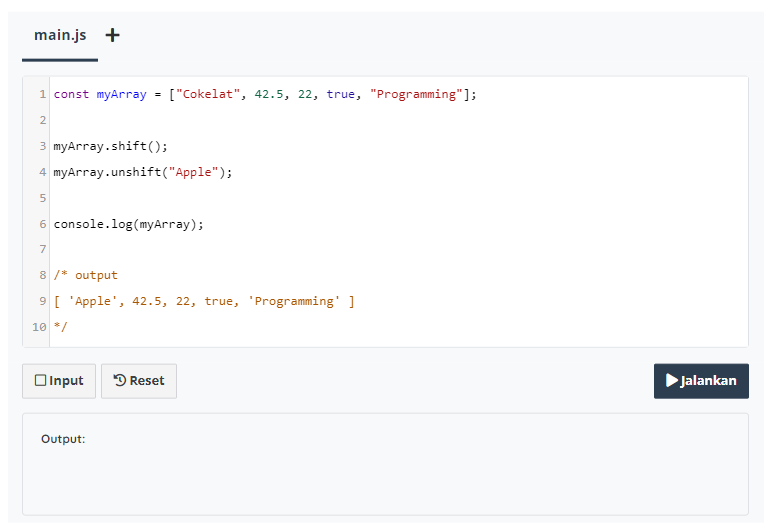
7

[ Cokelat, 42.5, 22, true ]

8

\*/

Metode lain yang bisa kita gunakan untuk memanipulasi data pada array adalah shift() dan unshift(). Metode shift() digunakan untuk mengeluarkan elemen pertama dari array, sementara unshift() digunakan untuk menambahkan elemen di awal array.



1

const myArray = ["Cokelat", 42.5, 22, true, "Programming"];

2

​

3

myArray.shift();

4

myArray.unshift("Apple");

5

​

6

console.log(myArray);

7

​

8

/\* output

9

[ 'Apple', 42.5, 22, true, 'Programming' ]

10

\*/

 Input  Reset

 Jalankan

Lalu bagaimana jika kita ingin menghapus data dari array? Sama seperti object, kita bisa menggunakan keyword delete.



1

const myArray = ["Cokelat", 42.5, 22, true, "Programming"];

2

​

3

delete myArray[1];

4

console.log(myArray);

5

​

6

/\* output

7

[ 'Cokelat', <1 empty item>, 22, true, 'Programming' ]

8

\*/

Namun, perhatikan di sini bahwa keyword delete hanya menghapus data pada index yang ditentukan lalu membiarkan posisi tersebut kosong. Untuk menghapus elemen, gunakan metode splice() seperti ini:



1

const myArray = ["Cokelat", 42.5, 22, true, "Programming"];

2

​

3

myArray.splice(2, 1); // Menghapus dari index 2 sebanyak 1 elemen

4

console.log(myArray);

5

​

6

/\* output

7

[ 'Cokelat', 42.5, true, 'Programming' ]

8

\*/

## Spread Operator

Masih terkait dengan array, ES6 memiliki fitur menarik untuk membantu pengelolaan array menjadi lebih mudah, yaitu spread operator.

Sesuai namanya “spread”, fitur ini digunakan untuk menyebarkan nilai array atau lebih tepatnya iterable object menjadi beberapa elemen. Spread operator dituliskan dengan tiga titik (...). Mari kita lihat contoh kode berikut:

1. const favorites = ["Seafood", "Salad", "Nugget", "Soup"];
3. console.log(favorites);
5. /\* output
6. [ 'Seafood', 'Salad', 'Nugget', 'Soup' ]
7. \*/

Pada kode tersebut hasil yang dicetak adalah sebuah array (ditunjukkan dengan tanda [ ]), karena memang kita mencetak nilai favorites itu sendiri. Nah, dengan menggunakan spread operator kita dapat menyebarkan nilai-nilai dalam array tersebut.



1

const favorites = ["Seafood", "Salad", "Nugget", "Soup"];

2

​

3

console.log(...favorites);

4

​

5

/\* output

6

Seafood Salad Nugget Soup

7

\*/

 Input  Reset

 Jalankan

Terlihat perbedaannya? Mengapa bisa demikian? Spread operator bekerja seperti meleburkan nilai array menjadi beberapa elemen sesuai panjang nilai array-nya. Sehingga jika kita menuliskan kode seperti ini:

1. console.log(...favorites);

Sama seperti kita menuliskan kode seperti ini:

1. console.log(favorites[0], favorites[1], favorites[2], favorites[3]);

Spread operator dapat digunakan untuk menggabungkan dua buah array ke dalam array baru. Jika tidak menggunakan spread operator ini maka hasilnya akan seperti ini:



1

const favorites = ["Seafood", "Salad", "Nugget", "Soup"];

2

const others = ["Cake", "Pie", "Donut"];

3

​

4

const allFavorites = [favorites, others];

5

​

6

console.log(allFavorites);

7

​

8

/\* output

9

[

10

[ 'Seafood', 'Salad', 'Nugget', 'Soup' ],

11

[ 'Cake', 'Pie', 'Donut' ]

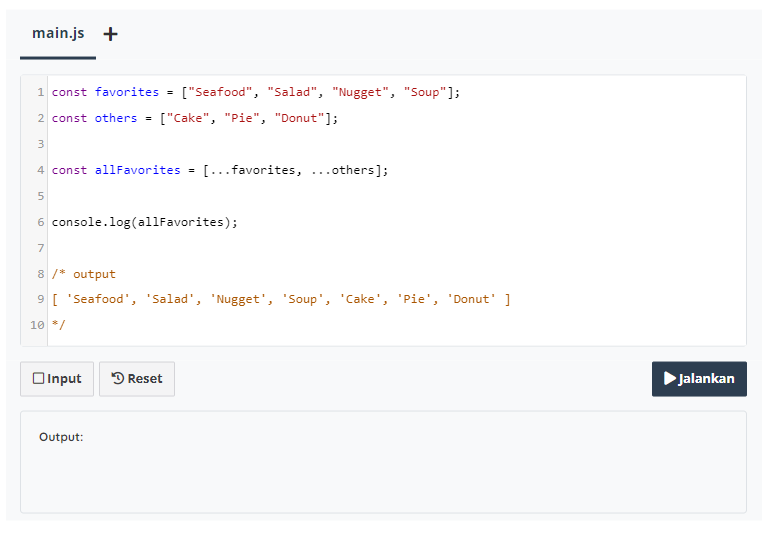
12

]

13

\*/

Nilai array tidak akan tergabung. Alih-alih menggabungkan nilainya, variabel allFavorites menjadi array baru yang menampung dua array di dalamnya. Nah, lantas bagaimana jika kita mencoba menggunakan spread operator?



1

const favorites = ["Seafood", "Salad", "Nugget", "Soup"];

2

const others = ["Cake", "Pie", "Donut"];

3

​

4

const allFavorites = [...favorites, ...others];

5

​

6

console.log(allFavorites);

7

​

8

/\* output

9

[ 'Seafood', 'Salad', 'Nugget', 'Soup', 'Cake', 'Pie', 'Donut' ]

10

\*/

Yup, dengan menggunakan spread operator nilai dua array tersebut berhasil tergabung.

Selain array, spread operator juga bisa digunakan untuk object literals. Hal ini memungkinkan kita dapat menggabungkan beberapa object dengan kode yang lebih ringkas.



1

const obj1 = { firstName: 'Obi-Wan', age: 30 };

2

const obj2 = { lastName: 'Kenobi', gender: 'M' };

3

​

4

const newObj = { ...obj1, ...obj2 };

5

​

6

console.log(newObj);

7

​

8

/\* output

9

{ firstName: 'Obi-Wan', age: 30, lastName: 'Kenobi', gender: 'M' }

10

\*/

## Destructuring Object & Array

Iterasi object dan array adalah dua hal yang paling banyak digunakan dalam mengelola data di JavaScript. JSON (JavaScript Object Notation) merupakan format data paling populer yang digunakan dalam transaksi data saat ini.

1. [
2. {
3. "id": 14,
4. "title": "Belajar Fundamental Aplikasi Android",
5. "author": "Google ATP"
6. },
7. {
8. "id": 51,
9. "title": "Belajar Membuat Aplikasi Android untuk Pemula",
10. "author": "Google ATP"
11. },
12. {
13. "id": 123,
14. "title": "Belajar Dasar Pemrograman Web",
15. "author": "Dicoding Indonesia"
16. },
17. {
18. "id": 163,
19. "title": "Belajar Fundamental Front-End Web Development",
20. "author": "Dicoding Indonesia"
21. }
22. ]

Jika kita lihat pada struktur JSON di atas, kita dapat menyimpulkan struktur tersebut dibangun dari array dan object. Karena kedua hal ini banyak digunakan untuk mengelola data pada JavaScript untuk memudahkan developer, ES6 menambahkan fitur untuk destructuring object dan array.

Apa sebenarnya destructuring object dan array itu? Destructuring dalam JavaScript merupakan sintaksis yang dapat mengeluarkan nilai dari array atau properties dari sebuah object ke dalam satuan yang lebih kecil.

Secara tidak sadar mungkin kita pernah melakukan destructuring. Namun, sebelum ES6 hal tersebut dilakukan dengan cara seperti ini:

* [**Object**](https://www.dicoding.com/academies/256/tutorials/15785?from=15775#tab1-code1)

1. const profile = {
2. firstName: "John",
3. lastName: "Doe",
4. age: 18
5. }
7. const firstName = profile.firstName
8. const lastName = profile.lastName
9. const age = profile.age
11. console.log(firstName, lastName, age)
13. /\* output:
14. John Doe 18
15. \*/

Perhatikan kode di atas, kode tersebut mengekstraksi nilai yang berada di dalam objek, kemudian menyimpannya pada variabel lokal dengan nama sama dengan properti di dalam object profile. Mungkin mengekstraksi nilai dari object dengan langkah ini terlihat mudah, tetapi bayangkan jika object memiliki banyak properti dan harus melakukan hal tersebut secara manual satu persatu. Terlalu banyak kode yang dituliskan berulang, bukan?

Itulah alasan ES6 menambahkan fitur yang memudahkan kita untuk destructuring object maupun array. Ketika kita ingin memecah struktur data menjadi bagian-bagian yang lebih kecil, kita akan dipermudah untuk mendapatkan data yang diinginkan.

Lantas bagaimana cara melakukan destructuring object dan array pada ES6? Mari kita simak materi berikutnya.

## Destructuring Object

Penulisan sintaksis destructuring object pada ES6 menggunakan object literal ({ }) di sisi kiri dari operator assignment.



1

const profile = {

2

firstName: "John",

3

lastName: "Doe",

4

age: 18

5

}

6

7

const { firstName, lastName, age } = profile;

8

9

console.log(firstName, lastName, age);

10

11

/\* output:

12

John Doe 18

13

\*/

Pada contoh di atas tanda kurung kurawal merepresentasikan object yang akan didestrukturisasi. Di dalamnya terdapat firstName, lastName, dan age yang merupakan variabel untuk menyimpan nilai properti dari object profile. Kita juga perlu perhatikan penamaan variabelnya. Pastikan penamaannya sama seperti properti object-nya. Melalui nama variabel inilah nilai-nilai properti object akan dimasukkan secara otomatis. Sehingga variabel firstName akan berisikan nilai profile.firstName, lastName akan berisikan nilai profile.lastName, begitu juga dengan variabel age akan berisikan nilai profile.age.

Dalam destructuring object, kita bisa menentukan salah satu nilai yang ingin kita desktrukturisasikan. Sehingga kita tidak perlu membuat variabel sebanyak properti yang dimiliki objeknya, contohnya:

1. const { lastName } = profile;

### Destructuring Assignment

Pada contoh sebelumnya, kita telah melakukan destructuring object pada deklarasi variabel. Namun, pada kasus tertentu mungkin kita perlu melakukannya pada variabel yang sudah dideklarasikan.



1

const profile = {

2

firstName: "John",

3

lastName: "Doe",

4

age: 18

5

}

6

7

let firstName = "Dimas";

8

let age = 20;

9

10

// menginisialisasi nilai baru melalui destructuring object

11

({ firstName, age } = profile);

12

13

console.log(firstName);

14

console.log(age);

15

16

/\* output:

17

John

18

18

19

\*/

Saat melakukan destructuring assignment, kita perlu menuliskan destructuring object di dalam tanda kurung. Jika tidak menuliskan tanda kurung, tanda kurung kurawal akan membuat JavaScript mengira kita membuat block statement, sementara block statement tidak bisa berada pada sisi kiri assignment.

1. // tidak bisa karena JavaScript mengira kita membuat block statement
2. // block statement tidak bisa berada pada sisi kiri assignment
3. { firstName, age } = profile;

Nah, inilah fungsinya tanda kurung. Ia akan memberi tahu JavaScript bahwa tanda kurawal di dalamnya bukan sebuah block statement melainkan sebuah expression, sehingga assignment dapat dilakukan.

1. ({ firstName, age } = profile);

### Default Values

Ketika kita mendestruksikan objek dan menetapkan variabel dengan nama yang bukan merupakan properti dari objek, maka nilai dari variabel tersebut menjadi undefined. Contohnya:



1

const profile = {

2

firstName: "John",

3

lastName: "Doe",

4

age: 18

5

}

6

7

8

const { firstName, age, isMale } = profile;

9

10

console.log(firstName)

11

console.log(age)

12

console.log(isMale)

13

14

/\* output:

15

John

16

18

17

undefined

18

\*/

Alternatifnya, kita bisa secara opsional mendefinisikan nilai default pada properti tertentu jika tidak ditemukan. Untuk melakukanya, tambahkan tanda assignment (=) setelah nama variabel dan tentukan nilai default-nya seperti ini:



1

const profile = {

2

firstName: "John",

3

lastName: "Doe",

4

age: 18

5

}

6

7

8

const { firstName, age, isMale = false } = profile;

9

10

console.log(firstName)

11

console.log(age)

12

console.log(isMale)

13

14

/\* output:

15

John

16

18

17

false

18

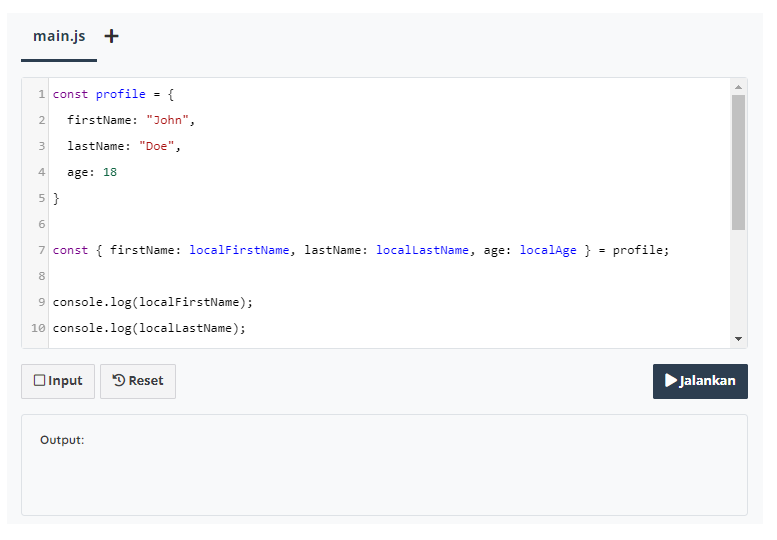
\*/

Jika nilai properti tidak ditemukan, maka nilai default akan diterapkan pada variabel.

### **Assigning to Different Local Variable Names**

Sampai saat ini kita tahu bahwa untuk melakukan destrukturisasi object pada variabel lokal, kita perlu menyeragamkan penamaan variabel lokal dengan properti object-nya. Namun, sebenarnya dalam proses destrukturisasi object kita bisa menggunakan penamaan variabel lokal yang berbeda. ES6 menyediakan sintaksis tambahan yang membuat kita dapat melakukan hal tersebut. Penulisannya mirip seperti ketika kita membuat properti beserta nilainya pada object.

Contohnya seperti ini:



1

const profile = {

2

firstName: "John",

3

lastName: "Doe",

4

age: 18

5

}

6

7

const { firstName: localFirstName, lastName: localLastName, age: localAge } = profile;

8

9

console.log(localFirstName);

10

console.log(localLastName);

11

console.log(localAge);

12

13

14

/\* output:

15

John

16

Doe

17

18

18

\*/

## Destructuring Array

Destructuring array serupa dengan destructuring object. Object menggunakan tanda kurung kurawal { } sedangkan array menggunakan tanda kurung siku [ ]. Perbedaan lainnya adalah destructuring array bekerja berdasarkan posisi daripada penamaan propertinya. Berikut contoh dari destructuring array pada ES6:



1

const favorites = ["Seafood", "Salad", "Nugget", "Soup"];

2

3

const [firstFood, secondFood, thirdFood, fourthFood] = favorites;

4

5

console.log(firstFood);

6

console.log(secondFood);

7

console.log(thirdFood);

8

console.log(fourthFood);

9

10

/\* output:

11

Seafood

12

Salad

13

Nugget

14

Soup

15

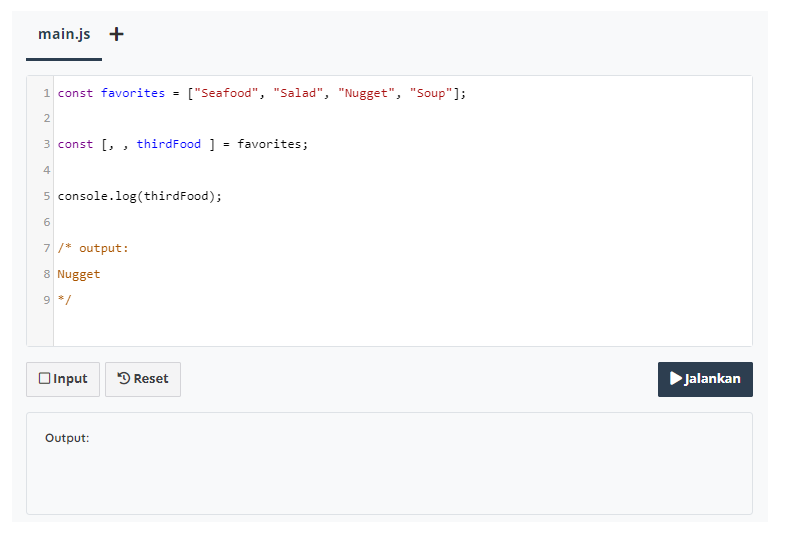
\*/

Kode di atas merupakan contoh proses destructuring array. Di dalam array favorites terdapat 4 (empat) nilai string yang masing-masing nilainya dimasukkan ke variabel lokal firstFood, secondFood, thirdFood, dan fourthFood. Nilai dari array yang dimasukkan ke variabel lokal dipilih berdasarkan posisi di mana ia dideklarasikan pada array.

1. const [firstFood, secondFood, thirdFood, fourthFood] = favorites;

Sebenarnya kita bebas untuk menentukan nama dari variabel lokal. Yang terpenting adalah urutan ketika deklarasi variabelnya saja.

Kita juga bisa memilih nilai pada index tertentu untuk destrukturisasi pada array. Contohnya, jika ingin mengambil nilai ketiga dari array, kita tidak perlu menyiapkan variabel lokal untuk menampung nilai array pertama, kedua, atau pun keempat. Kita bisa melakukannya dengan membiarkan index array yang tidak kita inginkan tetap kosong (tanpa menulis variabel lokal). Lebih lanjut, tanda koma (,) tetap diperlukan untuk menunjukkan posisi index-nya seperti ini:



1

const favorites = ["Seafood", "Salad", "Nugget", "Soup"];

2

3

const [, , thirdFood ] = favorites;

4

5

console.log(thirdFood);

6

7

/\* output:

8

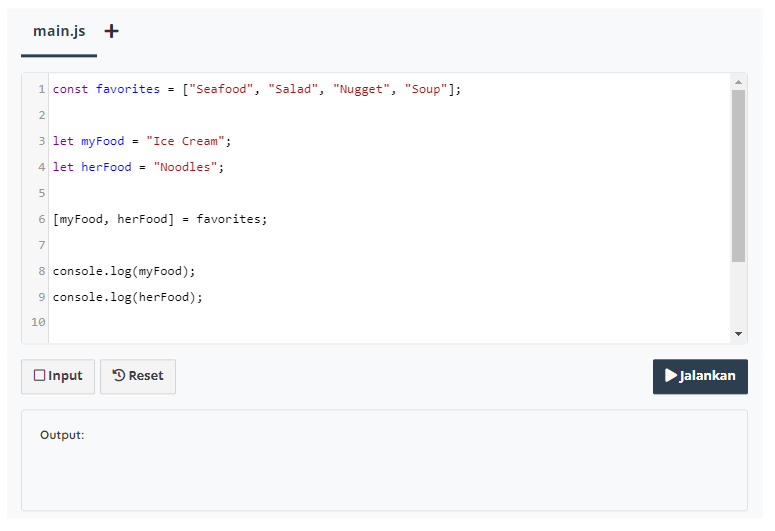
Nugget

9

\*/

### **Destructuring Assignment**

Kita juga bisa melakukan destructuring assignment pada array. Namun, tidak seperti object, kita tidak perlu membungkusnya dengan tanda kurung. Contohnya seperti berikut:



1

const favorites = ["Seafood", "Salad", "Nugget", "Soup"];

2

3

let myFood = "Ice Cream";

4

let herFood = "Noodles";

5

6

[myFood, herFood] = favorites;

7

8

console.log(myFood);

9

console.log(herFood);

10

11

/\* output:

12

Seafood

13

Salad

14

\*/

Array destructuring assignment sangat berguna ketika kita hendak menukar nilai antara dua variabel. Sebelum ES6, untuk melakukan hal ini kita menggunakan cara manual menggunakan algoritma sorting seperti ini:



1

var a = 1;

2

var b = 2;

3

var temp;

4

5

console.log("Sebelum swap");

6

console.log("Nilai a: " + a);

7

console.log("Nilai b: " + b);

8

9

temp = a;

10

a = b;

11

b = temp;

12

13

console.log("Setelah swap");

14

console.log("Nilai a: " + a);

15

console.log("Nilai b: " + b);

16

17

/\* output

18

Sebelum swap

19

Nilai a: 1

20

Nilai b: 2

Untuk melakukan pertukaran nilai, kita membutuhkan variabel penengah. Pada contoh kode di atas menggunakan variabel temp. Variabel penengah dibutuhkan untuk menyimpan data sementara pada variabel yang akan ditukar. Hal ini menjadi kurang efektif karena kita harus membuat variabel baru yang sebenarnya hanya bersifat sementara.

Dengan array destructuring assignment, kita bisa menukar nilai variabel dengan mudah tanpa membuat variabel tambahan.



1

let a = 1;

2

let b = 2;

3

4

console.log("Sebelum swap");

5

console.log("Nilai a: " + a);

6

console.log("Nilai b: " + b);

7

8

[a, b] = [b, a] // menetapkan nilai a dengan nilai b dan nilai b dengan nilai a.

9

10

console.log("Setelah swap");

11

console.log("Nilai a: " + a);

12

console.log("Nilai b: " + b);

13

14

/\* output

15

Sebelum swap

16

Nilai a: 1

17

Nilai b: 2

18

Setelah swap

19

Nilai a: 2

20

Nilai b: 1

### **Default Values**

Ketika melakukan destructuring array, tetapi terdapat variabel yang posisinya tidak dapat terjangkau oleh array, maka variabel tersebut akan bernilai undefined. Contohnya:

1. const favorites = ["Seafood"];
2. const [myFood, herFood] = favorites
4. console.log(myFood);
5. console.log(herFood);
7. /\* output:
8. Seafood
9. undefined
10. \*/

Sama seperti object, pada destructuring array kita juga dapat memberikan nilai default pada variabel yang tidak dapat terjangkau oleh array, sehingga nilai pada variabel tidak akan menjadi undefined.



1

const favorites = ["Seafood"];

2

3

const [myFood, herFood = "Salad"] = favorites

4

5

console.log(myFood);

6

console.log(herFood);

7

8

/\* output:

9

Seafood

10

Salad

11

\*/

## Map

Map adalah tipe data yang menyimpan koleksi data dengan format key-value layaknya Object. Yang membedakan adalah Map memperbolehkan key dengan tipe data apa pun, dibandingkan Object yang hanya mengizinkan key bertipe String atau Symbol.

Untuk mendefinisikan Map gunakan constructor seperti di bawah ini:

1. const myMap = new Map();

Apabila ingin menetapkan nilai dari Map secara langsung, gunakan array multi dimensi seperti ini:



1

const myMap = new Map([

2

['1', 'a String key'],

3

[1, 'a number key'],

4

[true, true]

5

]);

6

​

7

console.log(myMap);

8

​

9

/\* output

10

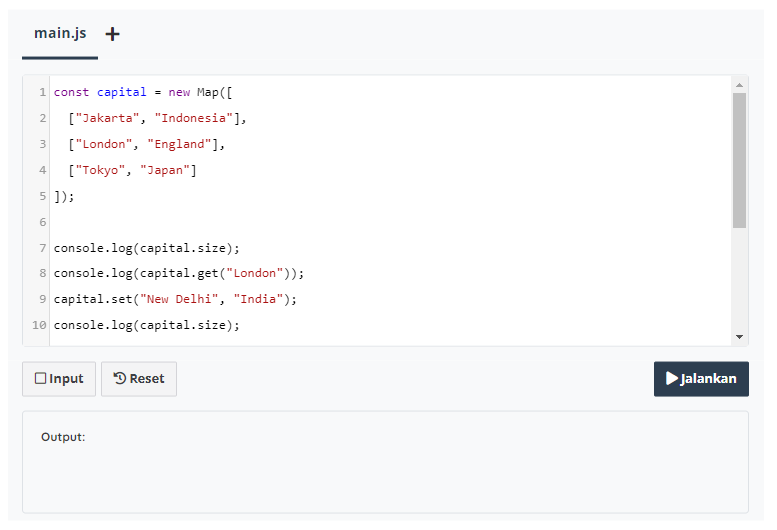
Map(3) { '1' => 'a String key', 1 => 'a number key', true => true }

11

\*/

Array pertama (yang berada di luar) berfungsi untuk menyimpan masing-masing elemen atau pasangan key-value dari Map. Kemudian array di dalamnya, memiliki dua elemen, di mana elemen pertama adalah key dan elemen keduanya merupakan value.

Ketika sudah membuat objek Map, kita bisa mendapatkan nilainya berdasarkan key tertentu dengan metode get(). Lalu, untuk menambahkan pasangan key-value baru gunakan metode set().



1

const capital = new Map([

2

["Jakarta", "Indonesia"],

3

["London", "England"],

4

["Tokyo", "Japan"]

5

]);

6

​

7

console.log(capital.size);

8

console.log(capital.get("London"));

9

capital.set("New Delhi", "India");

10

console.log(capital.size);

11

console.log(capital.get("New Delhi"));

12

​

13

/\* output

14

3

15

England

16

4

17

India

18

\*/

Note:  
Meskipun kita bisa menetapkan dan mendapatkan data pada map seperti ini:

1. const wrongMap = new Map();
3. wrongMap["My Key"] = "My Value";
4. console.log(wrongMap["My Key"]);

Namun, kode di atas bukanlah cara yang sesuai untuk mengoperasikan Map. Ketika kita menetapkan nilai map seperti di atas, data akan disimpan sebagai generic object. Ini akan mengakibatkan data tidak tersimpan dalam Map query dan tidak bisa menggunakan fitur dari Map seperti .has atau .delete.



1

const wrongMap = new Map();

2

wrongMap["My Key"] = "My Value";

3

​

4

console.log(wrongMap.has("My Key"));

5

console.log(wrongMap.delete("My Key"));

6

​

7

/\* output

8

false

9

false

10

\*/

Jadi, pastikan untuk menggunakan metode .set() dan .get() untuk Map.

## Set

Struktur data yang akan kita bahas berikutnya adalah Set. Set sederhananya merupakan kumpulan nilai (set of values). Hal yang membedakan Set dengan struktur data yang lain adalah data pada Set tidak berurutan dan juga tidak diindeks. Selain itu, data di dalam Set juga bersifat unik dan tidak ada duplikasi. Perhatikan contoh deklarasi Set di bawah ini:



1

const numberSet = new Set([1, 4, 6, 4, 1]);

2

​

3

console.log(numberSet);

4

​

5

/\* output

6

Set(3) { 1, 4, 6 }

7

\*/

Pada kode di atas terdapat beberapa angka yang duplikat, yaitu angka 1 dan 4. Secara otomatis Set akan membuang angka yang sama, sehingga nilai yang tersimpan adalah {1, 4, 6}.

Untuk menambahkan data ke dalam Set kita bisa memanfaatkan fungsi add().



1

const numberSet = new Set([1, 4, 6, 4, 1]);

2

numberSet.add(5);

3

numberSet.add(10);

4

numberSet.add(6);

5

​

6

console.log(numberSet);

7

​

8

/\* output

9

Set(5) { 1, 4, 6, 5, 10 }

10

\*/

Fungsi add() hanya menerima satu argumen. Jika Anda memasukkan array, maka array tersebut akan dianggap sebagai satu elemen sendiri. Nilai yang duplikat akan diabaikan.



1

const numberSet = new Set([1, 4, 6, 4, 1]);

2

numberSet.add(5);

3

numberSet.add(10);

4

numberSet.add(6);

5

​

6

numberSet.delete(4);

7

​

8

console.log(numberSet);

9

​

10

/\* output

11

Set(4) { 1, 6, 5, 10 }

12

\*/

 Input  Reset

 Jalankan

Ingat bahwa Set tidak memiliki urutan atau index, sehingga argumen yang dimasukkan ke dalam fungsi delete adalah nilai yang ingin dihapus, bukan index-nya.

## WeakMap & WeakSet

WeakMap merupakan varian dari Map yang mendukung garbage collection. Garbage collection adalah proses di mana interpreter JavaScript mengambil kembali memori yang tidak lagi “dapat dijangkau” dan tidak dapat digunakan oleh program [[3](https://learning.oreilly.com/library/view/javascript-the-definitive/9781491952016/ch11.html)]. Garbage collection di JavaScript dilakukan secara otomatis dan bukan menjadi urusan dari developer.

Yang dimaksud weak dalam WeakMap adalah referensi terhadap nilai yang disimpan. Apabila suatu nilai yang disimpan di WeakMap sudah tidak terjangkau atau tidak bisa lagi diakses, maka referensi ke memorinya akan dihapus.

Berikut ini adalah beberapa hal yang membedakan antara Map dan WeakMap:

* Key dari WeakMap harus berupa object atau array. Nilai primitif tidak bisa digunakan sebagai key karena tidak mendukung garbage collection.
* WeakMap memiliki method get(), set(), has(), dan delete(). Namun, WeakMap tidak termasuk kategori iterable sehingga tidak memiliki method keys(), values(), atau forEach().
* WeakMap juga tidak memiliki property size. Ini karena ukuran WeakMap dapat berubah karena proses garbage collection.

Masih bingung? Mari kita lihat contoh kode dan perbedaan antara Map dan WeakMap.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/15795#run1-editor1)



1

const visitsCountMap = new Map(); // Menyimpan daftar user

2

​

3

function countUser(user) {

4

let count = visitsCountMap.get(user) || 0;

5

visitsCountMap.set(user, count + 1);

6

}

7

​

8

let jonas = { name: "Jonas" };

9

countUser(jonas); // Menambahkan user "Jonas"

10

​

11

jonas = null; // Data object "Jonas" dihapus

12

​

13

// delay dibutuhkan untuk menunggu garbage collector bekerja

14

setTimeout(function() {

15

console.log(visitsCountMap);

16

}, 10000)

17

​

18

/\* output

19

Map(1) { { name: 'Jonas' } => 1 }

20

\*/

 Input  Reset

 Jalankan

setTimeout merupakan fungsi yang digunakan untuk menunda eksekusi kode yang ada di dalamnya hingga jangka waktu yang ditetapkan. Fungsi setTimeout akan detail dibahas pada modul concurrency.

Ketika reference objek jonas dihapus dengan mengubahnya menjadi null, seharusnya map tidak lagi menyimpan data user (garbage collected). Namun, kenyataannya data jonas masih tersedia di dalam Map. Artinya, data jonas masih tersimpan di dalam memori sampai kita benar-benar menghapusnya.

Berbeda jika kita menggunakan WeakMap seperti ini.

1. const visitsCountMap = new WeakMap();

Ketika nilai jonas sudah tidak bisa dijangkau, object jonas akan dihapus dari memori termasuk informasi yang disimpan di dalam WeakMap.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/15795#run2-editor1)



1

const { inspect } = require('util');

2

​

3

const visitsCountMap = new WeakMap(); // Menyimpan daftar user

4

​

5

function countUser(user) {

6

let count = visitsCountMap.get(user) || 0;

7

visitsCountMap.set(user, count + 1);

8

}

9

​

10

let jonas = { name: "Jonas" };

11

countUser(jonas); // Menambahkan user "Jonas"

12

​

13

jonas = null; // Data object "Jonas" dihapus

14

​

15

// delay dibutuhkan untuk menunggu garbage collector bekerja

16

setTimeout(function() {

17

console.log(inspect(visitsCountMap, { showHidden: true }));

18

}, 10000);

19

​

20

/\* output

 Input  Reset

 Jalankan

WeakMap tidak dapat dilihat secara langsung menggunakan console.log. Ini merupakan issue yang terjadi pada Node.js (lihat: <https://github.com/nodejs/node/issues/19001>). Itulah mengapa kami menggunakan fungsi [inspect](https://nodejs.org/api/util.html#utilinspectobject-options) untuk melihat data yang ada di dalam WeakMap.

Jika hasil output tidak sesuai dengan yang diharapkan pada kode, hal itu disebabkan oleh waktu tunggu yang tidak cukup untuk menunjukkan cara kerja garbage collector. Silakan simak penjelasan pada diskusi berikut: <https://www.dicoding.com/academies/256/discussions/156523>

Seperti halnya WeakMap, WeakSet adalah versi weak reference dari Set. Perbedaan antara WeakSet dan Set antara lain:

* WeakSet tidak bisa menyimpan nilai primitif.
* WeakSet bukan iterable dan hanya memiliki method add(), has(), dan delete().
* WeakSet tidak memiliki properti size.

## Kuis Coding : Object

Untuk menguji kemampuan praktikal Anda dalam memahami materi object di JavaScript, silakan selesaikan kuis berikut.

Sebelum Anda mengerjakan kuis, ada beberapa hal yang perlu Anda perhatikan.

* Mohon untuk membaca secara seksama perintah, kriteria, atau soal pada komentar kode yang diberi tanda **TODO.**
* Lakukan pengujian pada kode yang Anda tulis contohnya dengan menggunakan console.log() untuk memastikan hasil sesuai dengan yang Anda harapkan. Gunakan tombol **Jalankan**.

Bila sudah yakin dengan pekerjaan yang Anda lakukan, silakan submit kode dengan klik tombol **Submit kode**. Jika kode yang di-submit benar, Anda bisa melanjutkan ke modul berikutnya. Jika tidak, Anda bisa memperbaiki kembali kode sebanyak yang Anda mau.

Batas waktu tunggu untuk mengirimkan kembali perbaikan kode adalah 1 menit. Jadi pastikan kode yang Anda tulis sudah memenuhi perintah, kriteria, atau soal pada kuis kali ini.

Good luck!

|  |
| --- |
| **Mengalami kendala?** Cobalah kunjungi [Forum Diskusi](https://www.dicoding.com/academies/256/discussions?tutorial=15885) untuk mencari solusi atau bertanya langsung kepada mentor kami. |

Selamat, Anda telah menyelesaikan modul ini.  
Kode yang Anda kirimkan telah berfungsi dengan baik.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/15885?from=15795#run1-editor1)



1

/\*\*

2

\* TODO

3

\* 1. Buatlah variabel dengan nama restaurant yang bertipe object dengan ketentuan berikut:

4

\* - Memiliki properti bernama "name"

5

\* - Bertipe data string

6

\* - Bernilai apa pun, asalkan tidak string kosong atau null.

7

\* - Memiliki properti bernama "city"

8

\* - Bertipe data string

9

\* - Bernilai apa pun, asalkan tidak string kosong atau null.

10

\* - Memiliki properti "favorite drink"

11

\* - Bertipe data string

12

\* - Bernilai apa pun, asalkan tidak string kosong atau null.

13

\* - Memiliki properti "favorite food"

14

\* - Bertipe data string

15

\* - Bernilai apa pun, asalkan tidak string kosong atau null.

16

\* - Memiliki properti "isVegan"

17

\* - Bertipe data boolean

18

\* - Bernilai boolean apa pun.

19

\*

20

\* 2. Buatlah variabel bernama name.

21

\* Kemudian isi dengan nilai name dari properti object restaurant

22

\* 3. Buatlah variabel bernama favoriteDrink.

23

\* Kemudian isi dengan nilai "favorite drink" dari properti object restaurant

24

\*/

25

​

26

​

27

// TODO

28

let restaurant = {name: "bakso", city: "subang", "favorite drink": "es susu","favorite food": "mie ayam", isVegan: true};

29

let name = restaurant.name;

30

let favoriteDrink = restaurant["favorite drink"];

31

/\*\*

32

\* Jangan hapus kode di bawah ini

33

\*/

34

module.exports = { restaurant, name, favoriteDrink };

 Input  Reset

 Jalankan  Submit Kode

## Kuis Coding : Array

Untuk menguji kemampuan praktikal Anda dalam memahami materi Array di JavaScript, silakan selesaikan kuis berikut.

Sebelum Anda mengerjakan kuis, ada beberapa hal yang perlu Anda perhatikan.

* Mohon untuk membaca secara seksama perintah, kriteria, atau soal pada komentar kode yang diberi tanda **TODO.**
* Lakukan pengujian pada kode yang Anda tulis contohnya dengan menggunakan console.log() untuk memastikan hasil sesuai dengan yang Anda harapkan. Gunakan tombol **Jalankan**.

Bila sudah yakin dengan pekerjaan yang Anda lakukan, silakan submit kode dengan klik tombol **Submit kode**. Jika kode yang di-submit benar, Anda bisa melanjutkan ke modul berikutnya. Jika tidak, Anda bisa memperbaiki kembali kode sebanyak yang Anda mau.

Batas waktu tunggu untuk mengirimkan kembali perbaikan kode adalah 1 menit. Jadi pastikan kode yang Anda tulis sudah memenuhi perintah, kriteria, atau soal pada kuis kali ini.

Good luck!

|  |
| --- |
| **Mengalami kendala?** Cobalah kunjungi [Forum Diskusi](https://www.dicoding.com/academies/256/discussions?tutorial=15890) untuk mencari solusi atau bertanya langsung kepada mentor kami. |

Selamat, Anda telah menyelesaikan modul ini.  
Kode yang Anda kirimkan telah berfungsi dengan baik.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/15890#run1-editor1)



1

/\*\*

2

\* TODO:

3

\* Buatlah sebuah variabel dengan nama evenNumber yang merupakan sebuah array dengan ketentuan:

4

\* - Array tersebut menampung bilangan genap dari 1 hingga 100

5

\*

6

\* Catatan:

7

\* - Agar lebih mudah, gunakanlah for loop dan logika if untuk mengisi bilangan genap pada array.

8

\*/

9

​

10

// TODO

11

let evenNumber = [];

12

for(let i = 1; i <= 100; i++){

13

if(i % 2 == 0){

14

evenNumber.push(i);

15

}

16

}

17

​

18

/\*\*

19

\* Jangan hapus kode di bawah ini

20

\*/

21

​

22

module.exports = evenNumber;

23

​

 Input  Reset

 Jalankan  Submit Kode

## Kuis Coding : Map

Untuk menguji kemampuan praktikal Anda dalam memahami materi Map di JavaScript, silakan selesaikan kuis berikut.

Sebelum Anda mengerjakan kuis, ada beberapa hal yang perlu Anda perhatikan.

* Mohon untuk membaca secara seksama perintah, kriteria, atau soal pada komentar kode yang diberi tanda **TODO.**
* Lakukan pengujian pada kode yang Anda tulis contohnya dengan menggunakan console.log() untuk memastikan hasil sesuai dengan yang Anda harapkan. Gunakan tombol **Jalankan**.

Bila sudah yakin dengan pekerjaan yang Anda lakukan, silakan submit kode dengan klik tombol **Submit kode**. Jika kode yang di-submit benar, Anda bisa melanjutkan ke modul berikutnya. Jika tidak, Anda bisa memperbaiki kembali kode sebanyak yang Anda mau.

Batas waktu tunggu untuk mengirimkan kembali perbaikan kode adalah 1 menit. Jadi pastikan kode yang Anda tulis sudah memenuhi perintah, kriteria, atau soal pada kuis kali ini.

Good luck!

|  |
| --- |
| **Mengalami kendala?** Cobalah kunjungi [Forum Diskusi](https://www.dicoding.com/academies/256/discussions?tutorial=15900) untuk mencari solusi atau bertanya langsung kepada mentor kami. |

Selamat, Anda telah menyelesaikan modul ini.  
Kode yang Anda kirimkan telah berfungsi dengan baik.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/15900#run1-editor1)



1

/\*\*

2

\* TODO:

3

\* 1. Buatlah variabel currency yang merupakan Map dengan kriteria:

4

\* - key "USD", value 14000

5

\* - key "JPY", value 131

6

\* - key "SGD", value 11000

7

\* - key "MYR", value 3500

8

\* 2. Buatlah variabel priceInIDR yang bernilai dari hasil perkalian:

9

\* - priceInJPY dengan nilai currency JPY

10

\*/

11

​

12

const priceInJPY = 5000;

13

​

14

// TODO

15

const currency = new Map([

16

['USD', 14000],

17

['JPY', 131],

18

['SGD', 11000],

19

['MYR', 3500],

20

]);

 Input  Reset

 Jalankan  Submit Kode

## Rangkuman Materi

Untuk menyelesaikan masalah yang lebih kompleks secara efisien, kita membutuhkan struktur data yang lebih canggih dibandingkan variabel primitif. JavaScript memiliki tipe data lain yang dapat menyimpan banyak data sekaligus, contohnya seperti Object, Arrays, Map, dan Set.

Beberapa hal yang telah kita bahas pada modul ini, antara lain:

* Object adalah tipe data kompleks untuk menyimpan banyak data dengan format key-value. Key pada object harus berupa string.
* Array digunakan untuk menyimpan banyak data. Data pada array disusun secara berurutan dan diakses menggunakan index.
* Map menyimpan banyak data dengan format key-value seperti object. Namun, key pada Map dapat bertipe apa pun.
* Set digunakan untuk menyimpan banyak data secara unik, tidak ada duplikasi, tidak berurutan, dan tidak diindeks.

## Video Struktur Data di JavaScript

Untuk memperdalam dan mempermudah pamahaman pada materi ini, Anda dapat menyimak video pembahasan berikut.

<https://youtu.be/etDR8rc_vvs>

### Materi Pendukung

Berikut ini adalah beberapa sumber materi tambahan yang bisa Anda pelajari:

* [Object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object)
* [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array)
* [Map](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map)
* [Set](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set)
* [WeakMap and WeakSet](https://javascript.info/weakmap-weakset)
* [Garbage collection](https://javascript.info/garbage-collection)

Aturan

Knowledge Check : Struktur Data bertujuan untuk menguji pengetahuan Anda tentang materi yang telah dipelajari di modul ini.

Terdapat 2 pertanyaan yang harus dikerjakan dalam kuis ini. Beberapa ketentuannya sebagai berikut:

* Syarat nilai kelulusan : 100%
* Durasi ujian : 5 menit

Apabila tidak memenuhi syarat kelulusan, Anda bisa mengulang pengerjaan kuis kembali.

Selamat Mengerjakan!

# [Function](https://www.dicoding.com/academies/256/tutorials/15327?from=13912)

## Pengenalan Function

Pada modul ini salah satu konsep yang tidak kalah penting pada JavaScript, yaitu Function.

Di sini kita akan mempelajari beberapa hal terkait function, seperti:

* Apa itu function?
* Tujuan dan alasan kenapa kita menggunakan function.
* Bagaimana membuat function pada JavaScript.

Mari kita mulai!

## Declaring Function

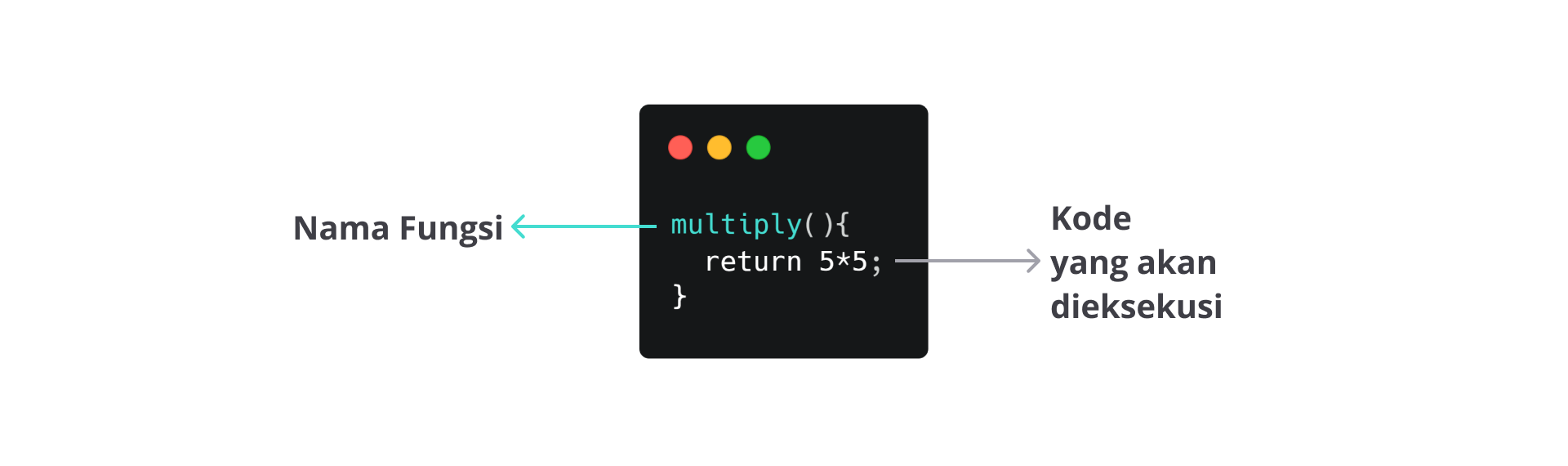
Fungsi merupakan bagian penting dalam bahasa pemrograman. Tanpa sadar, sebenarnya kita sudah menggunakan sebuah fungsi pada contoh kode yang ada sebelumnya. log() pada console.log() merupakan sebuah function yang berguna untuk menampilkan data pada konsol. Tapi sebenarnya apa itu function? Bagaimana ia bisa bekerja?

Mirip dengan fungsi pada matematika, fungsi dalam pemrograman juga digunakan untuk menghasilkan output berdasarkan input tertentu.

[](https://www.dicoding.com/academies/256/tutorials/13917?from=15327)

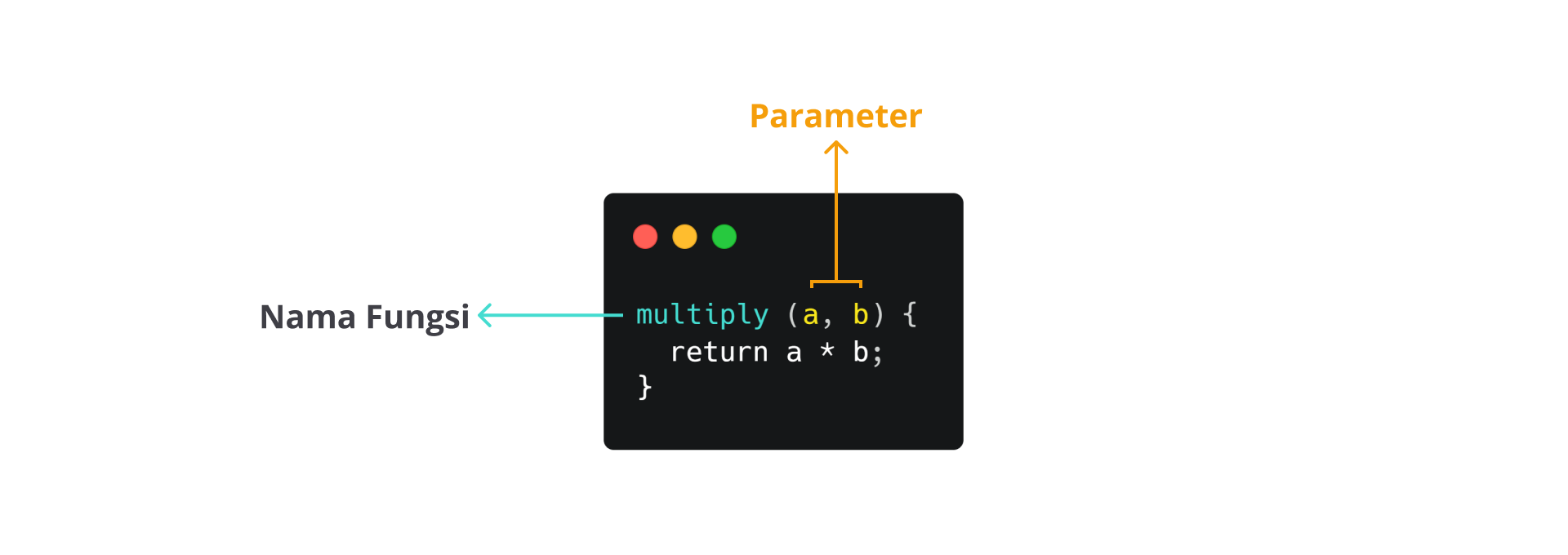
Namun, fungsi juga bisa digunakan sebagai blok kode atau prosedur yang dapat digunakan secara berulang. Dalam arti lain, kita dapat berpikir bahwa function merupakan sebuah variabel yang berisi blok logika. Blok logika tersebut akan dieksekusi ketika variabelnya dipanggil.

Semua fungsi memiliki struktur yang sama. Fungsi dideklarasikan dengan keyword function dan nama fungsinya. Nama fungsi selalu diikuti dengan tanda kurung (parentheses) tanpa spasi, lalu terdapat sepasang kurung kurawal yang berisi logika dari fungsi tersebut.

[](https://www.dicoding.com/academies/256/tutorials/13917?from=15327)

Terkadang di dalam tanda kurung kita membutuhkan sebuah informasi tambahan yang disebut dengan parameter. Parameter merupakan data yang digunakan pada fungsi untuk diproses di dalamnya. Sebagai contoh, fungsi console.log() dapat menerima argument berupa string atau data lain untuk ditampilkan ke konsol.

Berikut merupakan ilustrasi dari struktur fungsi dengan parameter:

[](https://www.dicoding.com/academies/256/tutorials/13917?from=15327)

**Parameter & Argument**

Di dalam fungsi kita akan banyak bertemu istilah parameter & argument. Penggunaan istilah ini sering kali tertukar, bahkan di kalangan developer.

 Perbedaan mendasar antara keduanya antara lain:

Parameter merupakan variabel yang didefinisikan sebagai inputan dari sebuah fungsi. Contoh:

1. function multiply(**a, b**) {
2. return a \* b;
3. }

Argument merupakan nilai atau expression yang dimasukkan ke dalam fungsi. Contohnya:

1. multiply(**3, 4**);

Setelah membuat fungsi kita dapat memanggilnya dengan menuliskan nama fungsi diikuti tanda kurung dan memasukkan argumen di dalamnya (jika ada).

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13917?from=15327#run1-editor1)



1

function greeting() {

2

console.log("Good Morning!")

3

}

4

​

5

greeting();

6

​

7

/\* output

8

Good Morning!

9

\*/

 Input  Reset

 Jalankan

Tetapi jika sebuah fungsi hanya menjalankan baris kode yang sama dirasa kurang fungsional, bukan? Kita dapat membuat fungsi tersebut untuk menerima dan memanfaatkan parameter untuk mengubah perilaku dari fungsinya.

Untuk menambahkan parameter pada fungsi, tambahkan variabel di dalam tanda kurung fungsi. Namun, variabel tersebut tidak memerlukan keyword var, let, ataupun const. Kita juga bisa menambahkan lebih dari satu parameter dengan memberikan tanda koma antar variabel parameternya. Contohnya fungsi greeting akan kita tambahkan parameter name dan language seperti ini:

1. function greeting(name, language) {
2. if(language === "English") {
3. console.log(`Good Morning ${name}!`);
4. } else if (language === "French") {
5. console.log(`Bonjour ${name}!`);
6. } else {
7. console.log(`Selamat Pagi ${name}!`);
8. }
9. }

Sehingga dalam memanggilnya pun kita perlu mengirimkan dua buah nilainya sebagai argumen:

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13917?from=15327#run2-editor1)



1

function greeting(name, language) {

2

if(language === "English") {

3

console.log(`Good Morning ${name}!`);

4

} else if (language === "French") {

5

console.log(`Bonjour ${name}!`);

6

} else {

7

console.log(`Selamat Pagi ${name}!`);

8

}

9

}

10

​

11

greeting("Harry", "French");

12

​

13

/\* output

14

Bonjour Harry!

15

\*/

 Input  Reset

 Jalankan

Satu hal lagi, function dapat menghasilkan output atau mengembalikan sebuah nilai. Dengan nilai kembalian, kita dapat membuat function yang berfungsi untuk melakukan perhitungan matematika dan hasilnya dapat kita masukkan ke dalam sebuah variabel. Contohnya seperti ini:

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13917?from=15327#run3-editor1)



1

function multiply(a, b) {

2

return a \* b;

3

}

4

​

5

let result = multiply(10, 2)

6

console.log(result)

7

​

8

/\* output

9

20

10

\*/

 Input  Reset

 Jalankan

Agar fungsi bisa mengembalikan nilai, gunakan keyword return diikuti dengan nilai yang akan dikembalikan. Nilai kembalian tidak hanya number, bisa juga berupa string, boolean, object, array, atau tipe yang lain. Seperti inilah fungsi greeting() jika kita ubah agar mengembalikan nilai string:

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13917?from=15327#run4-editor1)



1

function greeting(name, language) {

2

if(language === "English") {

3

return `Good Morning ${name}!`

4

} else if (language === "French") {

5

return `Bonjour ${name}!`;

6

} else {

7

return `Selamat Pagi ${name}!`;

8

}

9

}

10

​

11

let greetingMessage = greeting("Harry", "French");

12

console.log(greetingMessage);

13

​

14

/\* output

15

Bonjour Harry!

16

\*/

 Input  Reset

 Jalankan

Yang perlu kita perhatikan lagi, ketika statement return tereksekusi, maka fungsi akan langsung terhenti dan mengembalikan nilai.

### Expression Function

Cara lain untuk membuat sebuah fungsi pada JavaScript adalah expression function. Ingat kembali bahwa expression adalah kode atau instruksi yang mengembalikan nilai, sehingga expression function bisa disimpan dalam sebuah variabel.

Pada expression function umumnya kita tidak perlu menuliskan nama fungsinya. Fungsi yang tidak bernama juga dikenal dengan anonymous function. Berikut ini merupakan contoh penulisan expression function:

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13917?from=15327#run5-editor1)



1

const greeting = function(name, language) {

2

if(language === "English") {

3

return "Good Morning " + name + "!";

4

} else if (language === "French") {

5

return "Bonjour " + name + "!";

6

} else {

7

return "Selamat Pagi " + name + "!";

8

}

9

}

10

​

11

console.log(greeting('Ron', 'English'));

12

​

13

/\* output

14

Good Morning Ron!

15

\*/

 Input  Reset

 Jalankan

## Function Parameter

Pada materi sebelumnya, kita telah berkenalan dengan fungsi yang merupakan suatu blok kode yang dapat menerima input dan menghasilkan output tertentu. Input ke dalam sebuah fungsi dikirimkan melalui parameter.

Di dalam fungsi kita akan banyak bertemu istilah parameter & argument. Penggunaan istilah ini sering kali tertukar, bahkan di kalangan developer.

Perbedaan mendasar antara keduanya antara lain:

Parameter merupakan variabel yang didefinisikan sebagai inputan dari sebuah fungsi. Contoh:

1. function multiply(a, b) {
2. return a \* b;
3. }

Argument merupakan nilai atau expression yang dimasukkan ke dalam fungsi. Contohnya:

1. multiply(3, 4);

Parameter dari fungsi dapat berupa tipe data apa pun, mulai dari string, number, object, bahkan fungsi lain.

Jika parameter dari fungsi adalah sebuah object, kita juga bisa memanfaatkan destructuring object untuk mendapatkan nilainya. Contohnya seperti berikut:

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/17272?from=13917#run1-editor1)



1

const user = {

2

id: 24,

3

displayName: 'kren',

4

fullName: 'Kylo Ren',

5

};

6

​

7

function introduce({displayName, fullName}) {

8

console.log(`${displayName} is ${fullName}`);

9

}

10

​

11

introduce(user);

12

​

13

/\* output

14

kren is Kylo Ren

15

\*/

 Input  Reset

 Jalankan

### Default Parameters

Fungsi pada JavaScript tidak melakukan pengecekan terhadap jumlah maupun tipe argumen yang dimasukkan ke dalam parameter. Ini berarti kita bisa memasukkan argumen meskipun tidak sesuai dengan parameter yang telah didefinisikan.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/17272?from=13917#run2-editor1)



1

function exponentsFormula(baseNumber, exponent) {

2

const result = baseNumber \*\* exponent;

3

console.log(`${baseNumber}^${exponent} = ${result}`);

4

}

5

​

6

exponentsFormula(2);

7

​

8

/\* output

9

2^undefined = NaN

10

\*/

 Input  Reset

 Jalankan

Seperti yang kita lihat pada contoh kode di atas, ketika argumen lebih sedikit dari parameternya, maka parameter yang tidak terdefinisi akan bernilai undefined. Sebagai solusi jika memungkinkan, kita bisa memberikan nilai default pada parameter. Nilai ini akan digunakan jika kita tidak memasukkan parameter.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/17272?from=13917#run3-editor1)



1

function exponentsFormula(baseNumber, exponent = 2) {

2

const result = baseNumber \*\* exponent;

3

console.log(`${baseNumber}^${exponent} = ${result}`);

4

}

5

​

6

exponentsFormula(3);

7

​

8

/\* output

9

3^2 = 9

10

\*/

 Input  Reset

 Jalankan

### Rest Parameter

Masih ingat dengan spread operator (...)? Jika spread operator menyebarkan array menjadi beberapa elemen berbeda, rest parameter ini adalah kebalikan dari operator tersebut. Penasaran?

Rest parameter juga dituliskan menggunakan three consecutive dots (...). Dengan rest parameter, kita dapat menggabungkan beberapa elemen menjadi satu array. Tentu teknik ini sangat bermanfaat ketika kita hendak membuat sebuah fungsi dengan parameter yang tidak pasti.

Sebagai contoh adalah fungsi yang menjumlahkan seluruh nilai argumen seperti berikut:

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/17272?from=13917#run4-editor1)



1

function sum(...numbers) {

2

let result = 0;

3

for (let number of numbers) {

4

result += number;

5

}

6

return result;

7

}

8

​

9

console.log(sum(1, 2, 3, 4, 5));

10

​

11

/\* output

12

15

13

\*/

 Input  Reset

 Jalankan

## Arrow Function

ES6 memperkenalkan fungsi baru yang dinamakan arrow function expression atau lebih dikenal sebagai arrow function. Arrow function mirip seperti regular function secara perilaku, tetapi berbeda dalam penulisannya. Sesuai namanya, fungsi didefinisikan menggunakan tanda panah atau fat arrow ( => ). Tentunya penulisan arrow function ini akan lebih singkat.

Selain perbedaan sintaksis, terdapat perbedaan perilaku antara arrow function dan regular function. Regular function dapat berupa function declaration dan function expression. Namun, arrow function hanya berupa expression function saja. Itu sebabnya arrow function memiliki nama lengkap “arrow function expression”.

* [**Regular function**](https://www.dicoding.com/academies/256/tutorials/13922?from=17272#tab1-code1)
* [Arrow function](https://www.dicoding.com/academies/256/tutorials/13922?from=17272#tab1-code2)

1. // function declaration
2. function sayHello(greet) {
3. console.log(`${greet}!`);
4. }
6. // function expression
7. const sayName = function (name) {
8. console.log(`Nama saya ${name}`)
9. }

Pada arrow function kita tidak perlu menuliskan keyword function setiap membuat fungsi. Kita tetap menuliskan parameter di dalam tanda kurung lalu diikuti dengan tanda panah (=>) sebelum kurung kurawal.

1. const sayName = (name) => {
2. console.log(`Nama saya ${name}`)
3. }

Apabila fungsi hanya memiliki satu parameter, maka kita bisa menghapuskan tanda kurung seperti berikut:

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13922?from=17272#run1-editor1)



1

const sayName = name => {

2

console.log(`Nama saya ${name}`)

3

}

4

​

5

sayName("Leia");

6

​

7

/\* output

8

Nama saya Leia

9

\*/

 Input  Reset

 Jalankan

Namun, jika kita sama sekali tidak membutuhkan parameter, maka kita tetap menuliskan tanda kurung namun kosong seperti ini:

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13922?from=17272#run2-editor1)



1

const sayHello = () => {

2

console.log("Selamat pagi semuanya!")

3

};

4

​

5

sayHello();

6

​

7

/\* output

8

Selamat pagi semuanya!

9

\*/

 Input  Reset

 Jalankan

Satu hal yang menarik, ketika body dari function hanya terdiri dari satu baris, kita bisa menghapus tanda kurung kurawal. Tentunya ini akan menghemat baris kode yang kita tulis.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13922?from=17272#run3-editor1)



1

const sayName = name => console.log(`Nama saya ${name}`);

2

sayName("Leia");

3

​

4

const sayHello = () => console.log("Selamat pagi semuanya!");

5

sayHello();

6

​

7

/\* output

8

Nama saya Leia

9

Selamat pagi semuanya!

10

\*/

 Input  Reset

 Jalankan

Ketika sebuah fungsi perlu mengembalikan nilai, kita tidak perlu lagi menuliskan return (hanya bekerja untuk fungsi satu baris).

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13922?from=17272#run4-editor1)



1

const multiply = (a, b) => a \* b;

2

console.log(multiply(3, 4));

3

​

4

/\* output

5

12

6

\*/

 Input  Reset

 Jalankan

## Variable Scope

Sejauh ini kita sudah mengenal function. Setelah kita memisahkan kode ke dalam blok atau fungsi terpisah, ada satu hal penting yang perlu kita tahu, yaitu variable scoping. Ada banyak keadaan di mana kita membutuhkan variabel untuk diakses di seluruh script yang kita buat. Tetapi ada juga keadaan di mana kita ingin variabel tersebut hanya dapat diakses pada cakupan fungsi dan fungsi turunannya saja.

Variabel yang dapat diakses dari seluruh script disebut dengan “globally scoped”, sementara variabel yang hanya diakses hanya pada fungsi tertentu disebut dengan “locally scoped”.

Variabel JavaScript menggunakan fungsi untuk mengelola cakupannya. Jika variabel didefinisikan di luar fungsi, maka variabel tersebut bersifat global. Jika variabel didefinisikan di dalam fungsi, maka variabel bersifat lokal dan cakupannya hanya pada fungsi tersebut beserta turunannya.

Berikut ini merupakan contoh scoping dalam kode:

1. // global variable, dapat diakses pada parent() dan child()
2. const a = 'a';
4. function parent() {
5. // local variable, dapat diakses pada parent() dan child(), tetapi tidak dapat diakses di luar dari fungsi tersebut.
6. const b = 'b';
8. function child() {
9. // local variable, dapat diakses hanya pada fungsi child().
10. const c = 'c';
11. }
12. }

Kita harus berhati-hati dalam mendefinisikan variabel di dalam fungsi. Pasalnya, kita bisa mendapatkan hasil yang tidak diperkirakan, contohnya seperti berikut:

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13927?from=13922#run1-editor1)



1

function multiply(num) {

2

total = num \* num;

3

return total;

4

}

5

​

6

let total = 9;

7

let number = multiply(20);

8

​

9

console.log(total)

10

​

11

/\* output

12

400

13

\*/

 Input  Reset

 Jalankan

Mungkin kita berharap nilai total akan tetap 9, mengingat variabel total pada fungsi multiply seharusnya tidak akan berpengaruh untuk kode di luar dari fungsi tersebut. Hal ini bisa terjadi karena pada fungsi multiply() kita tidak menetapkan variabel total sebagai cakupan lokal. Kita tidak menggunakan keyword const atau let ketika mendeklarasikan variabel total pada fungsi multiply() sehingga variabel total menjadi global.

Perlu kita perhatikan, jika kita lupa menuliskan keyword let, const, atau var pada script ketika membuat sebuah variabel, maka variabel tersebut akan menjadi global.

Sebisa mungkin kita harus menghindari pembuatan variabel global, karena variabel global dapat diakses pada seluruh script yang kita tuliskan. Semakin banyak variabel global yang kita tuliskan, semakin tinggi kemungkinan tabrakan (collision) terjadi.

## Closure

Setelah mempelajari tentang scope pada materi sebelumnya, kali ini kita akan membahas seputar closure. Sebelumnya kita telah tahu bahwa fungsi dapat didefinisikan dalam lingkup global atau di dalam fungsi lain. Suatu fungsi yang dapat mengakses variabel di dalam lexical scope-nya disebut dengan closure. Lexical scope berarti pada sebuah fungsi bersarang, fungsi yang berada di dalam memiliki akses ke variabel di lingkup induknya.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13928?from=13927#run1-editor1)



1

function init() {

2

const name = 'Obi Wan'; // Variabel lokal di dalam scope fungsi init

3

4

function greet() { // Inner function, merupakan contoh closure

5

console.log(`Halo, ${name}`); // Memanggil variabel yang dideklarasikan di parent function

6

}

7

​

8

greet();

9

}

10

​

11

init();

12

​

13

/\* output

14

Halo, Obi Wan

15

\*/

 Input  Reset

 Jalankan

Fungsi init() memiliki variabel lokal name dan fungsi greet(). Fungsi greet() adalah inner function yang didefinisikan di dalam init() dan hanya bisa diakses dari dalam fungsi init(). Perhatikan bahwa fungsi greet() tidak memiliki variabel lokal. Namun, karena inner function memiliki akses ke variabel di parent function-nya, sehingga greet() dapat mengakses variabel name. Itulah yang dimaksud dengan lexical scope.

Sekarang perhatikan contoh kode berikut:

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13928?from=13927#run2-editor1)



1

function init() {

2

const name = 'Obi Wan';

3

​

4

function greet() {

5

console.log(`Halo, ${name}`);

6

}

7

​

8

return greet;

9

}

10

​

11

const myFunction = init();

12

myFunction();

13

​

14

/\* output

15

Halo, Obi Wan

16

\*/

 Input  Reset

 Jalankan

Kode di atas akan menghasilkan output yang sama. Perbedaannya adalah fungsi greet() dikembalikan (return) dari outer function-nya sebelum dieksekusi. Karena variabel name berada dalam scope init(), maka umumnya variabel tersebut akan hilang atau dihapus ketika fungsinya selesai dijalankan. Namun, pada kasus di atas fungsi greet() yang diakses melalui fungsi MyFunction() masih memiliki referensi atau akses ke variabel name. Variabel pada mekanisme di atas telah tertutup (close covered), yang berarti variabel tersebut berada di dalam closure.

Memang di awal cukup sulit untuk memahami closure. Jadi, mari kita lihat langsung untuk apa closure ini digunakan pada suatu program yang nyata.

JavaScript tidak memiliki cara untuk mendeklarasikan suatu fungsi atau variabel menjadi private seperti bahasa Java. Sehingga sebuah fungsi atau variabel bisa diakses dari mana pun. Kenapa kita membutuhkan private method? Salah satunya adalah untuk membatasi akses ke fungsi atau variabel. Perhatikan contoh berikut:

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13928?from=13927#run3-editor1)



1

let counter = 0;

2

​

3

const add = () => {

4

return ++counter;

5

}

6

​

7

console.log(add());

8

console.log(add());

9

counter = 23;

10

console.log(add());

11

​

12

/\* output

13

1

14

2

15

24

16

\*/

 Input  Reset

 Jalankan

Nilai counter akan bertambah ketika kita memanggil fungsi add(). Namun, kita juga bisa mengubah nilai counter secara langsung dengan assignment operator. Pada contoh program yang lebih kompleks, sebaiknya hal ini dihindari karena perubahan langsung pada nilai counter bisa saja memunculkan bug.

Closure memungkinkan kita membuat fungsi dan variabel seolah menjadi private. Seperti inilah contoh program counter yang dibuat dengan closure:

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13928?from=13927#run4-editor1)



1

const add = () => {

2

let counter = 0;

3

return () => {

4

return ++counter;

5

};

6

}

7

​

8

const addCounter = add();

9

​

10

console.log(addCounter());

11

console.log(addCounter());

12

console.log(addCounter());

13

​

14

/\* output

15

1

16

2

17

3

18

\*/

 Input  Reset

 Jalankan

## Kuis Coding : Function

Untuk menguji kemampuan praktikal Anda dalam memahami materi Function di JavaScript, silakan selesaikan kuis berikut.

Sebelum Anda mengerjakan kuis, ada beberapa hal yang perlu Anda perhatikan.

* Mohon untuk membaca secara seksama perintah, kriteria, atau soal pada komentar kode yang diberi tanda **TODO.**
* Lakukan pengujian pada kode yang Anda tulis contohnya dengan menggunakan console.log() untuk memastikan hasil sesuai dengan yang Anda harapkan. Gunakan tombol **Jalankan**.

Bila sudah yakin dengan pekerjaan yang Anda lakukan, silakan submit kode dengan klik tombol **Submit kode**. Jika kode yang di-submit benar, Anda bisa melanjutkan ke modul berikutnya. Jika tidak, Anda bisa memperbaiki kembali kode sebanyak yang Anda mau.

Batas waktu tunggu untuk mengirimkan kembali perbaikan kode adalah 1 menit. Jadi pastikan kode yang Anda tulis sudah memenuhi perintah, kriteria, atau soal pada kuis kali ini.

Good luck!

|  |
| --- |
| **Mengalami kendala?** Cobalah kunjungi [Forum Diskusi](https://www.dicoding.com/academies/256/discussions?tutorial=15905) untuk mencari solusi atau bertanya langsung kepada mentor kami. |

Selamat, Anda telah menyelesaikan modul ini.  
Kode yang Anda kirimkan telah berfungsi dengan baik.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/15905#run1-editor1)



23

​

24

// TODO

25

function minimal(a, b) {

26

if (a <= b) {

27

return a;

28

} else {

29

return b;

30

}

31

}

32

console.log(minimal(1, 4));

33

console.log(minimal(3, 2));

34

console.log(minimal(3, 3));

35

​

36

function power(a, b){

37

return Math.pow(a, b)

38

}

39

/\*\*

40

\* Jangan hapus kode di bawah ini

41

\*/

42

​

43

module.exports = { minimal, power };

44

​

 Input  Reset

 Jalankan  Submit Kode

## Rangkuman Materi

Untuk membuat kode yang bersih dan mudah dipahami, salah satu cara yang bisa kita lakukan adalah dengan menggunakan function.

Beberapa hal yang telah kita bahas pada modul ini, antara lain:

* Function/fungsi adalah blok atau kumpulan kode yang memungkinkan untuk dipanggil berkali-kali.
* Penamaan fungsi harus mendeskripsikan fungsi dengan jelas. Penamaan yang bagus akan langsung memberi kita pemahaman tentang apa yang dilakukan atau dikembalikan.
* Function dapat menerima masukan nilai melalui parameter.
* Fungsi dapat mengembalikan nilai. Jika tidak, maka hasilnya adalah undefined.
* Karena fungsi berupa nilai, fungsi dapat ditetapkan atau dideklarasikan di setiap tempat kode, termasuk variabel atau parameter dari fungsi lain.

## Video Function di JavaScript

Untuk memperdalam dan mempermudah pemahaman dalam materi ini, Anda dapat menyimak video pembahasan berikut.

<https://youtu.be/AL9ipPNzSOE>

### Materi Pendukung:

Berikut ini adalah beberapa sumber materi tambahan yang bisa Anda pelajari:

* [Function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function)
* [Function expression](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/function)
* [Arrow function expressions](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions)
* [Closures](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures)

Aturan

Knowledge Check : Function bertujuan untuk menguji pengetahuan Anda tentang materi yang telah dipelajari di modul ini.

Terdapat 2 pertanyaan yang harus dikerjakan dalam kuis ini. Beberapa ketentuannya sebagai berikut:

* Syarat nilai kelulusan : 100%
* Durasi ujian : 5 menit

Apabila tidak memenuhi syarat kelulusan, Anda bisa mengulang pengerjaan kuis kembali.

Selamat Mengerjakan!

# [Object-Oriented Programming (OOP)](https://www.dicoding.com/academies/256/tutorials/15870?from=13932)

## Pengantar Object-Oriented Programming di JavaScript

Object-oriented programming atau disingkat OOP merupakan paradigma pemrograman yang banyak diterapkan ketika membangun aplikasi. Paradigma OOP kental dengan konsep objek yang mengandung dua hal, yaitu atribut dan method. Selain itu, OOP juga kental dengan konsep object blueprint yang akrab dikenal dengan sebutan class.

Meskipun JavaScript bukan bahasa pemrograman yang berbasis class seperti Java, tetapi di JavaScript kita juga bisa menerapkan paradigma OOP.

Di modul ini kita akan membahas beberapa hal terkait OOP, seperti:

* Pengertian paradigma Object-Oriented Programming.
* Constructor function dan sintaks class di JavaScript.
* Properti dan Method di dalam class.
* Pewarisan.
* Constructor dan method overriding.
* Object composition.
* Built-in class.

## Object-Oriented Programming

Object-Oriented Programming (OOP) adalah salah satu paradigma dalam pemrograman yang berfokus pada pembuatan sebuah objek dan interaksi dengan objek-objek tersebut. Di OOP, objek merupakan sebuah entitas yang terdiri dari dua hal, yakni properties dan methods.

* Properti merupakan nilai di dalam objek yang menyimpan informasi tentang objek tersebut.
* Method merupakan fungsi yang menggambarkan aksi yang dapat dilakukan oleh objek tersebut.

Paradigma OOP kerap digambarkan dengan kehidupan dunia nyata. Objek di dalam OOP, bisa kita anggap seperti objek yang ada di dunia nyata. Kita ambil contoh, mobil. Dalam OOP, sebuah mobil dapat dilihat sebagai objek yang memiliki beberapa properti seperti merek, warna, kecepatan maksimal, dan nomor rangka. Objek tersebut memiliki kemampuan atau method seperti maju, mundur, dan belok.

|  |  |
| --- | --- |
| Object | Mobil |
| Properties | Merek |
| Warna |
| Kecepatan maksimal |
| Nomor rangka |
| Methods | Maju |
| Mundur |
| Belok |

Lalu, seperti apa bentuk objek mobil dalam bentuk JavaScript? Sama seperti yang sudah Anda ketahui di modul Struktur Data, objek mobil dibuat dengan menggunakan tanda kurung kurawal dan untuk method-nya, Anda cukup buat properti yang merupakan sebuah fungsi, contohnya seperti ini.

1. const car = {
2. // properties
3. brand: 'Ford',
4. color: 'red',
5. maxSpeed: 200,
6. chassisNumber: 'f-1',
7. // methods
8. drive: () => {
9. console.log('driving');
10. },
11. reverse: () => {
12. console.log('reversing');
13. },
14. turn: () => {
15. console.log('turning');
16. }
17. }
19. console.log(car.brand); // Ford
20. console.log(car.color); // red
21. console.log(car.maxSpeed); // 200
22. console.log(car.chassisNumber); // f-1
23. car.drive(); // driving
24. car.reverse(); // reversing
25. car.turn(); // turning

Kode di atas merupakan contoh objek mobil atau car di JavaScript. Seperti yang Anda lihat, objek car memiliki properti brand, color, maxSpeed, dan chassisNumber; dan juga method drive, reverse, dan turn. Kita bisa akses nilai properti dan panggil method yang ada di dalam objek tersebut.

Di JavaScript untuk membuat sebuah objek terlihat mudah, bukan? Namun, masalah yang dipecahkan oleh paradigma OOP tidak hanya sebatas membuat objek sederhana saja. Seiring berkembangnya aplikasi, pembuatan objek akan semakin banyak dan saling berinteraksi satu dengan yang lainnya. Sehingga, kita harus mengetahui cara efektif mengelola objek termasuk cara membuatnya.

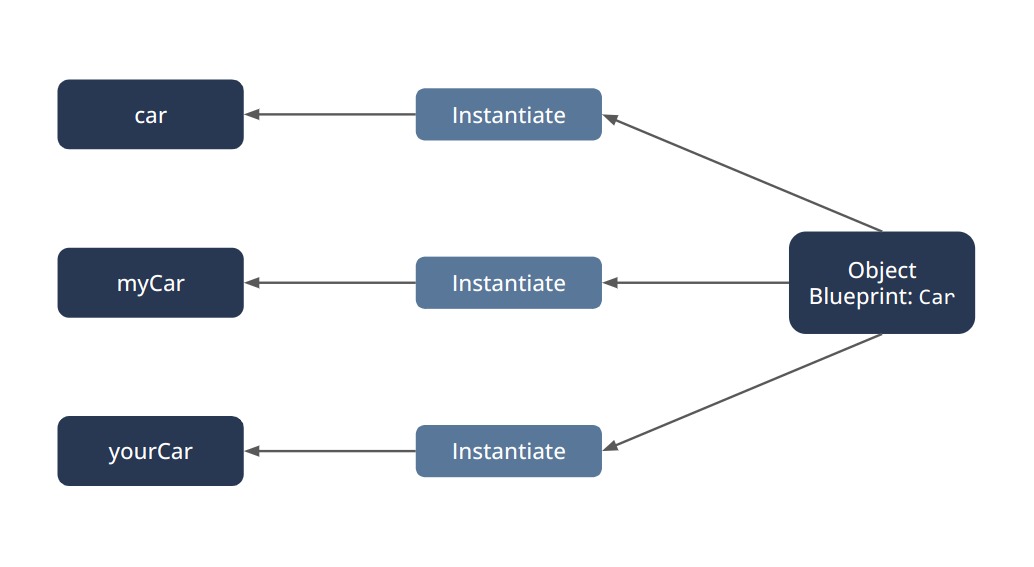
Mari kita angkat sebuah masalah baru dari contoh kode di atas. Bagaimana jika Anda ingin membuat objek dua mobil baru dengan nilai yang berbeda? Haruskah Anda mendefinisikan properti dan method yang sama secara berulang seperti contoh kode di bawah ini?

1. const car = {
2. brand: 'Ford',
3. color: 'red',
4. maxSpeed: 200,
5. chassisNumber: 'f-1',
6. drive: () => {
7. console.log('driving');
8. },
9. reverse: () => {
10. console.log('reversing');
11. },
12. turn: () => {
13. console.log('turning');
14. }
15. }
17. const myCar = {
18. brand: 'Tesla',
19. color: 'black',
20. maxSpeed: 250,
21. chassisNumber: 't-1',
22. drive: () => {
23. console.log('driving');
24. },
25. reverse: () => {
26. console.log('reversing');
27. },
28. turn: () => {
29. console.log('turning');
30. }
31. }
33. const yourCar = {
34. brand: 'BMW',
35. color: 'white',
36. maxSpeed: 300,
37. chassisNumber: 'b-1',
38. drive: () => {
39. console.log('driving');
40. },
41. reverse: () => {
42. console.log('reversing');
43. },
44. turn: () => {
45. console.log('turning');
46. }
47. }

Kami rasa kode di atas sungguh tidak efektif. Bagaimana jika ada banyak mobil yang harus Anda buat? Contoh, Anda membangun aplikasi yang memiliki entitas pelanggan. Jika Anda menggunakan cara di atas, mampukah Anda membuat objek pelanggan jika aplikasi sudah digunakan oleh 100 pelanggan?

Dalam memecahkan masalah ini, OOP menawarkan sebuah solusi yakni dengan membuat object blueprint. Melalui object blueprint, kita bisa membuat cetakan untuk membuat objek yang sudah terdefinisikan macam-macam properti dan method-nya. Sehingga kita cukup menggunakan cetakan tersebut untuk membuat objek yang serupa, tetapi kita bisa menentukan nilai-nilai properti yang berbeda.

Contohnya, Anda bisa membuat sebuah blueprint bernama Car. Di dalam blueprint tersebut, Anda bisa definisikan properti-properti dan method yang ingin dimiliki objek mobil nantinya. Setelah membuat sebuah blueprint, Anda bisa dengan mudah membuat banyak objek mobil contohnya myCar, yourCar, dan dicodingCar dengan lebih mudah.

[](https://www.dicoding.com/academies/256/tutorials/15880?from=15870)

Di dalam OOP, object blueprint tersebut bernama class. Penasaran seperti apa class di JavaScript? Simak materi selanjutnya, ya!

## Constructor Function

Kebanyakan paradigma OOP diterapkan pada [class-based programming languages](https://link.springer.com/chapter/10.1007/978-1-4419-8598-9_3), seperti Java, C++, C#, dan bahasa pemrograman berbasis class lainnya. JavaScript sendiri bukanlah bahasa pemrograman berbasis class karena sebelum hadirnya sintaks ES6, JavaScript tidak mengenal class. Namun, walaupun demikian, konsep-konsep yang ada di OOP masih bisa diterapkan di JavaScript.

JavaScript bukanlah class-based language, melainkan [prototype-based language](https://en.wikipedia.org/wiki/Prototype-based_programming). Menariknya, konsep-konsep OOP yang memanfaatkan class seperti pewarisan dapat dilakukan dengan memanfaatkan prototype. Alih-alih menggunakan class, prototype-lah yang diadaptasi JavaScript.

Kita akan membahas konsep pewarisan di beberapa materi mendatang, semantara kita fokus ke masalah sebelumnya, yaitu object blueprint.

Pertanyaannya, bagaimana cara membuat object blueprint di JavaScript? Fitrahnya, object blueprint dapat dibuat dengan menggunakan constructor function yang tampak seperti ini.

1. function Car(brand, color, maxSpeed, chassisNumber) {
2. this.brand = brand;
3. this.color = color;
4. this.maxSpeed = maxSpeed;
5. this.chassisNumber = chassisNumber;
6. }
8. Car.prototype.drive = function() {
9. console.log(`${this.brand} ${this.color} is driving`);
10. }
12. Car.prototype.reverse = function() {
13. console.log(`${this.brand} ${this.color} is reversing`);
14. }
16. Car.prototype.turn = function() {
17. console.log(`${this.brand} ${this.color} is turning`);
18. }

Kode di atas merupakan contoh membuat object blueprint dari sebuah mobil. Anda bisa membuat objek mobil--yang merupakan instance dari Car--dengan memanfaatkan constructor function Car.

1. function Car(brand, color, maxSpeed, chassisNumber) {
2. this.brand = brand;
3. this.color = color;
4. this.maxSpeed = maxSpeed;
5. this.chassisNumber = chassisNumber;
6. }
8. Car.prototype.drive = function() {
9. console.log(`${this.brand} ${this.color} is driving`);
10. }
12. Car.prototype.reverse = function() {
13. console.log(`${this.brand} ${this.color} is reversing`);
14. }
16. Car.prototype.turn = function() {
17. console.log(`${this.brand} ${this.color} is turning`);
18. }
20. // Membuat objek mobil dengan constructor function Car
21. const car1 = new Car('Toyota', 'Silver', 200, 'to-1');
22. const car2 = new Car('Honda', 'Black', 180, 'ho-1');
23. const car3 = new Car('Suzuki', 'Red', 220, 'su-1');
25. console.log(car1);
26. console.log(car2);
27. console.log(car3);
29. car1.drive();
30. car2.drive();
31. car3.drive();
33. /\* Output
34. Car { brand: 'Toyota', color: 'Silver', maxSpeed: 200, chassisNumber: 'to-1' }
35. Car { brand: 'Honda', color: 'Black', maxSpeed: 180, chassisNumber: 'ho-1' }
36. Car { brand: 'Suzuki', color: 'Red', maxSpeed: 220, chassisNumber: 'su-1' }
38. Toyota Silver is driving
39. Honda Black is driving
40. Suzuki Red is driving
41. \*/

Mungkin saat ini Anda bertanya-tanya, bagaimana bisa membuat sebuah objek dengan sebuah fungsi? Itulah salah satu kemampuan fungsi di JavaScript. Fungsi di atas bukanlah fungsi biasa, melainkan constructor function. Berikut beberapa perbedaan yang harus Anda perhatikan.

1. Penamaan fungsi yang merupakan constructor function, secara konvensi harus ditulis dengan huruf kapital. Itulah alasan kami memberi nama fungsi tersebut Car, bukan car. Hal ini penting untuk Anda ikuti guna membedakan fungsi biasa dengan constructor function karena penggunaannya berbeda.
2. Constructor function dapat memanfaatkan keywordthis yang bernilai objek (instance) dirinya sendiri. Keywordthis dapat dimanfaatkan untuk mengakses nilai properti atau method dari objek tersebut. Contoh, kami menggunakan keyword this untuk menetapkan nilai properti brand dari argumen fungsi. Selain itu, di dalam method drive, kami juga menggunakan this untuk mendapatkan nilai properti brand dan color.
3. Constructor function memiliki internal property bernama prototype. Properti ini digunakan untuk mendefinisikan method-method yang akan dimiliki oleh objek yang dibuat. Alasan method perlu didefinisikan di dalam prototype agar mudah untuk diwarisi ketika melakukan pewarisan.
4. Terakhir, agar fungsi mengembalikan sebuah objek, Anda harus memanggilnya dengan menambahkan keyword new. Contoh, pada kode di atas, perhatikan cara kami memanggil fungsi Car untuk membuat objek car1, car2, dan car3.

Perlu Anda ingat bahwa constructor function hanya dapat dibuat dengan reguler function. Anda tidak dapat membuat constructor function dengan arrow function. Arrow function tidak dapat dipanggil dengan keyword new.

1. const Car = (brand) => this.brand = brand;
3. const car1 = new Car('Toyota');
5. /\* Output
6. TypeError: Car is not a constructor
7. \*/

### Sintaks Class di ES6

Sintaks JavaScript modern (ES6) menawarkan cara membuat constructor function dengan menggunakan keyword class. Hal ini membuat penerapan OOP di JavaScript mirip seperti bahasa pemrograman berbasis class. Kita bisa menulis constructor function Car dengan gaya class seperti ini.

1. class Car {
2. constructor(brand, color, maxSpeed, chassisNumber) {
3. this.brand = brand;
4. this.color = color;
5. this.maxSpeed = maxSpeed;
6. this.chassisNumber = chassisNumber;
7. }
9. drive() {
10. console.log(`${this.brand} ${this.color} is driving`);
11. }
13. reverse() {
14. console.log(`${this.brand} ${this.color} is reversing`);
15. }
17. turn() {
18. console.log(`${this.brand} ${this.color} is turning`);
19. }
20. }
22. // Membuat objek mobil dengan constructor function Car
23. const car1 = new Car('Toyota', 'Silver', 200, 'to-1');
24. const car2 = new Car('Honda', 'Black', 180, 'ho-1');
25. const car3 = new Car('Suzuki', 'Red', 220, 'su-1');
27. console.log(car1);
28. console.log(car2);
29. console.log(car3);
31. car1.drive();
32. car2.drive();
33. car3.drive();
35. /\* Output
36. Car { brand: 'Toyota', color: 'Silver', maxSpeed: 200, chassisNumber: 'to-1' }
37. Car { brand: 'Honda', color: 'Black', maxSpeed: 180, chassisNumber: 'ho-1' }
38. Car { brand: 'Suzuki', color: 'Red', maxSpeed: 220, chassisNumber: 'su-1' }
40. Toyota Silver is driving
41. Honda Black is driving
42. Suzuki Red is driving
43. \*/

Jika Anda memiliki pengalaman di bahasa pemrograman C++, Java, atau C#, pembuatan class seperti di atas tentu lebih mudah dipahami, bukan? Memang inilah tujuan diciptakan sintaks class di JavaScript. Ia hadir untuk mempermudah penerapan OOP bagi developer yang memiliki latar belakang class-based language.

Walaupun di JavaScript sudah hadir sintaks class, hal itu tidak membuat JavaScript menjadi class-based language. Faktanya, sintaks class di JavaScript hanyalah syntactic sugar atau cara alternatif dalam mendefinisikan constructor function. Untuk membuktikan hal tersebut, kita bisa mengecek tipe class melalui operator typeof.

1. class Car {};
2. console.log(typeof Car);
4. /\* Output
5. function
6. \*/

Sekarang Anda sudah mengetahui cara membuat object blueprint di JavaScript dengan constructor function dan sintaks class. Di materi selanjutnya kita akan membahas mengenai properti dan method yang ada di dalam sebuah class atau constructor function.

**Catatan:** Anda sudah mengetahui bahwa ada dua cara dalam membuat function constructor, yakni dengan keyword function dan class. Di materi selanjutnya kami hanya fokus menggunakan sintaks class agar memudahkan pemahaman.

## Properti dan Method

Di materi sebelumnya, Anda sudah tahu bahwa class (function constructor dalam JavaScript) merupakan sebuah object blueprint yang dapat membuat sebuah objek serupa lebih mudah. Dengan menggunakan class, kita bisa terhindar dari banyak duplikasi kode dalam membuat banyak objek sekaligus.

Di dalam sebuah class, kita dapat mendefinisikan dua hal, yaitu properti dan method. Di materi kali ini kita akan membahas lebih detail mengenai keduanya.

### Properti

Properti merupakan bagian dari class yang mendefinisikan nilai-nilai yang terkandung dalam sebuah instance class. Contohnya, jika Anda membuat class Car, properti adalah informasi yang sekiranya terdapat pada sebuah mobil seperti brand, color, maxSpeed, dan chasissNumber. Contoh lain, jika Anda membuat class Mail, secara umum propertinya adalah sender, receiver, subject, dan body.

1. class Car {
2. constructor(brand, color, maxSpeed, chassisNumber) {
3. this.brand = brand;
4. this.color = color;
5. this.maxSpeed = maxSpeed;
6. this.chassisNumber = chassisNumber;
7. }
8. }
10. class Mail {
11. constructor(sender, receiver, subject, body) {
12. this.sender = sender;
13. this.receiver = receiver;
14. this.subject = subject;
15. this.body = body;
16. }
17. }

Nilai dari properti biasanya diambil dari dari argumen constructor agar nilainya dapat bervariasi setiap kali membuat instance.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/15895?from=13933#run1-editor1)



1

class Car {

2

constructor(brand, color, maxSpeed, chassisNumber) {

3

this.brand = brand;

4

this.color = color;

5

this.maxSpeed = maxSpeed;

6

this.chassisNumber = chassisNumber;

7

}

8

}

9

10

const car1 = new Car('BMW', 'red', 200, 'b-1');

11

const car2 = new Car('Audi', 'blue', 220, 'a-1');

12

const car3 = new Car('BMW', 'black', 250, 'b-2');

13

​

14

console.log(car1);

15

console.log(car2);

16

console.log(car3);

17

18

/\* Output:

19

Car { brand: 'BMW', color: 'red', maxSpeed: 200, chassisNumber: 'b-1' }

20

Car { brand: 'Audi', color: 'blue', maxSpeed: 220, chassisNumber: 'a-1' }

 Input  Reset

 Jalankan

Namun, adakalanya nilai properti juga bisa didefinisikan di dalam class itu sendiri. Contohnya, kita tidak ingin pengguna menentukan nomor rangka mobil secara mandiri, maka kita bisa memberi nilai properti chassisNumber langsung di dalam fungsi constructor.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/15895?from=13933#run2-editor1)



1

class Car {

2

constructor(brand, color, maxSpeed) {

3

this.brand = brand;

4

this.color = color;

5

this.maxSpeed = maxSpeed;

6

// Set a random chassis number

7

this.chassisNumber = `${brand}-${Math.floor(Math.random() \* 1000) + 1}`;

8

}

9

}

10

11

const car1 = new Car('BMW', 'red', 200);

12

const car2 = new Car('Audi', 'blue', 220);

13

const car3 = new Car('BMW', 'black', 250);

14

15

console.log(car1);

16

console.log(car2);

17

console.log(car3);

18

19

/\* Output:

20

Car { brand: 'BMW', color: 'red', maxSpeed: 200, chassisNumber: 'BMW-558' }

 Input  Reset

 Jalankan

#### Properti Getter dan Setter

Secara standar, properti di dalam sebuah instance class bersifat mutable atau dapat diubah nilainya. Meskipun kita sudah menetapkan nilai chassisNumber oleh sistem, nyatanya nilai tersebut dapat diubah dengan mudah ketika objek mobil telah dibuat.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/15895?from=13933#run3-editor1)



1

class Car {

2

constructor(brand, color, maxSpeed) {

3

this.brand = brand;

4

this.color = color;

5

this.maxSpeed = maxSpeed;

6

this.chassisNumber = `${brand}-${Math.floor(Math.random() \* 1000)}`;

7

}

8

}

9

10

const car = new Car('BMW', 'red', 200);

11

car.chassisNumber = 'BMW-1';

12

13

console.log(car);

14

15

/\* Output:

16

Car { brand: 'BMW', color: 'red', maxSpeed: 200, chassisNumber: 'BMW-1' }

17

\*/

 Input  Reset

 Jalankan

Lalu, bagaimana cara memproteksi agar nilai dari properti chassisNumber tidak dapat diubah? Nah, ketika kita berhadapan dengan kasus seperti ini, kita bisa memanfaatkan properti getter dan setter.

Sebelum memecahkan masalah di atas, ketahuilah bahwa ada dua tipe properti, yaitu **data property** dan **accessor property**.

Data property merupakan properti yang sejauh ini kita lihat, properti yang langsung menampung sebuah nilai di dalam sebuah objek.

Sedangkan accessor property merupakan properti yang dikontrol oleh sebuah getter dan setter. Nilai yang didapatkan dari properti tersebut dikontrol oleh method get dan cara menetapkan nilai tersebut dikontrol oleh method set.

Berikut contoh dari accessor property.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/15895?from=13933#run4-editor1)



1

class User {

2

constructor(firstName, lastName) {

3

this.firstName = firstName;

4

this.lastName = lastName;

5

}

6

7

get fullName() {

8

return `${this.firstName} ${this.lastName}`;

9

}

10

11

set fullName(fullName) {

12

const [firstName, lastName] = fullName.split(' ');

13

this.firstName = firstName;

14

this.lastName = lastName;

15

}

16

}

17

18

const user = new User('John', 'Doe');

19

console.log(user);

20

console.log(user.fullName);

 Input  Reset

 Jalankan

Di dalam class User, Anda bisa melihat bahwa terdapat data property firstName dan lastName. Nilai dari properti tersebut ditetapkan via argumen constructor. Selain itu, Anda juga bisa melihat sebuah method get fullName dan set fullname. Method tersebut merupakan accessor property yang mengatur cara akses dari properti fullName.

Sebab kita menetapkan getter dan setter untuk properti fullName, maka kita bisa mengakses properti tersebut melalui instance User. Ketika kita coba mendapatkan nilai properti fullName dengan cara user.fullName, method getter akan dijalankan dan nilai yang dikembalikan akan menjadi nilai dari properti tersebut. Begitu juga ketika kita coba menetapkan nilai properti fullName dengan cara user.fullName = “Fulan Fulanah”, kode di dalam method setter akan dijalankan.

Catatan penting yang perlu Anda ketahui mengenai getter setter adalah:

1. method getter harus mengembalikan sebuah nilai dan nilai tersebut akan menjadi nilai properti;
2. method setter harus menerima satu argumen yang nilainya diambil dari operand ke dua ketika melakukan assignment operator.

Mari kita kembali ke masalah perubahan nilai properti chassisNumber. Bagaimana getter dan setter dapat memproteksi perubahan properti chassisNumber? Di JavaScript, pola yang sering diterapkan untuk memecahkan masalah ini adalah dengan memanfaatkan getter setter sebagai “wrapper” dari properti aslinya. Tujuannya agar getter setter bisa mengontrol akses seperti mendapatkan dan menetapkan nilai properti.

Untuk menerapkan pola ini, pertama kita perlu mengubah nama dari properti chassisNumber, misalnya dengan menambahkan tanda garis bawah di depannya menjadi \_chassisNumber.

1. class Car {
2. constructor(brand, color, maxSpeed) {
3. this.brand = brand;
4. this.color = color;
5. this.maxSpeed = maxSpeed;
6. this.\_chassisNumber = `${brand}-${Math.floor(Math.random() \* 1000)}`;
7. }
8. }

Lalu, kita tetapkan getter dan setter untuk properti chassisNumber. Untuk getter, kita kembalikan dengan nilai properti \_chassisNumber.

1. class Car {
2. constructor(brand, color, maxSpeed) {
3. this.brand = brand;
4. this.color = color;
5. this.maxSpeed = maxSpeed;
6. this.\_chassisNumber = `${brand}-${Math.floor(Math.random() \* 1000)}`;
7. }
9. **get chassisNumber() {**
10. **return this.\_chassisNumber;**
11. **}**
12. }

Kita tidak ingin nilai chassisNumber dapat diubah sehingga untuk setter properti chassisNumber, cetak saja teks peringatan menggunakan console.log() seperti ini.

1. class Car {
2. constructor(brand, color, maxSpeed) {
3. this.brand = brand;
4. this.color = color;
5. this.maxSpeed = maxSpeed;
6. this.\_chassisNumber = `${brand}-${Math.floor(Math.random() \* 1000)}`;
7. }
9. get chassisNumber() {
10. return this.\_chassisNumber;
11. }
13. set chassisNumber(chassisNumber) {
14. console.log('you are not allowed to change the chassis number');
15. }
16. }

Lantas, ketika kita membuat instance Car, nilai chassisNumber tidak dapat diubah.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/15895?from=13933#run5-editor1)



1

class Car {

2

constructor(brand, color, maxSpeed) {

3

this.brand = brand;

4

this.color = color;

5

this.maxSpeed = maxSpeed;

6

this.\_chassisNumber = `${brand}-${Math.floor(Math.random() \* 1000)}`;

7

}

8

9

get chassisNumber() {

10

return this.\_chassisNumber;

11

}

12

13

set chassisNumber(chassisNumber) {

14

console.log('you are not allowed to change the chassis number');

15

}

16

}

17

18

const car = new Car('BMW', 'red', 200);

19

console.log(car.chassisNumber);

20

car.chassisNumber = 'BMW-1';

 Input  Reset

 Jalankan

|  |
| --- |
| **Catatan**  Sebenarnya nilai chassisNumber masih bisa berubah jika kita mengubah langsung melalui properti \_chassisNumber. Namun, ketahuilah bahwa mengubah atau mendapatkan nilai properti objek yang diawali dengan tanda underscore tidak direkomendasikan. Alasanya, komunitas JavaScript menyepakati bahwa hal properti yang diberi tanda underscore bukan untuk diakses, alias bersifat privat.  JavaScript versi ES2022 hadir dengan fitur private identifier. Dengan fitur tersebut, kita bisa membuat private property. Ini akan memecahkan masalah di atas. Private property akan kami bahas pada materi Member Visibility. |

### Method

Method adalah sebuah fungsi yang berada di dalam sebuah class dan dapat diakses melalui instance Class tersebut. Method biasanya mengindikasikan hal yang dapat dilakukan oleh sebuah class. Bila kita berbicara tentang class Car, method yang ada bisa drive(), reverse(), dan turn(). Jika pada class Mail, method bisa berupa send(), sendLater(), saveAsDraft().

1. class Car {
2. constructor(brand, color, maxSpeed) {
3. this.brand = brand;
4. this.color = color;
5. this.maxSpeed = maxSpeed;
6. this.\_chassisNumber = `${brand}-${Math.floor(Math.random() \* 1000)}`;
7. }
9. get chassisNumber() {
10. return this.\_chassisNumber;
11. }
13. set chassisNumber(chassisNumber) {
14. console.log('you are not allowed to change the chassis number');
15. }
17. // Methods
18. drive() {
19. console.log(`${this.brand} ${this.color} is driving`);
20. }
22. reverse() {
23. console.log(`${this.brand} ${this.color} is reversing`);
24. }
26. turn(direction) {
27. console.log(`${this.brand} ${this.color} is turning ${direction}`);
28. }
29. }
31. class Mail {
32. constructor(sender, receiver, subject, body) {
33. this.sender = sender;
34. this.receiver = receiver;
35. this.subject = subject;
36. this.body = body;
37. }
39. // Methods
40. send() {
41. console.log(`Sending mail from ${this.sender} to ${this.receiver}`);
42. }
44. sendLater(delay) {
45. console.log(`Sending after ${delay} ms`);
47. setTimeout(() => {
48. this.send();
49. }, delay);
50. }
52. saveAsDraft() {
53. console.log('Saving mail as draft');
54. }
55. }

Sama seperti fungsi JavaScript, method bisa menerima sebuah argumen. Contohnya pada method turn() di class Car dan sendLater di class Mail, kami memanfaatkan argumen direction dan delay untuk menetapkan arah dan waktu delay dalam menjalankan method-nya.

Selain argumen, method juga dapat memiliki akses ke nilai properti atau method lainnya melalui keyword this.

Method di dalam class hanya bisa dijalankan melalui instance dari class tersebut.

1. const car = new Car('BMW', 'red', 200);
3. car.drive();
4. car.turn('left');
5. car.reverse();
7. /\* Output:
8. BMW red is driving
9. BMW red is turning left
10. BMW red is reversing
11. \*/

Method memang kental dengan “kemampuan” yang bisa dilakukan oleh sebuah class, tetapi praktik nyatanya membuat method tidak hanya untuk itu. Method juga biasa dibuat ketika kita perlu mengekstraksi sebuah kode agar lebih mudah untuk dibaca dan method tersebut hanya digunakan untuk kebutuhan internal saja.

Contoh, pada class Car, kita menetapkan nilai \_chassisNumber dengan nilai random yang ditulis di dalam constructor. Hal itu membuat kode di dalam constructor menjadi sulit dibaca karena dicampuri dengan logika dalam menghasilkan angka acak.

Agar kode di dalam constructor lebih rapi, kita bisa membuat method yang digunakan internal yang menampung kode random tersebut. Biasanya method yang hanya digunakan secara internal disebut dengan private dan namanya diawali dengan tanda underscore.

1. class Car {
2. constructor(brand, color, maxSpeed) {
3. this.brand = brand;
4. this.color = color;
5. this.maxSpeed = maxSpeed;
6. this.\_chassisNumber = this.\_generateChassisNumber();
7. }
9. get chassisNumber() {
10. return this.\_chassisNumber;
11. }
13. set chassisNumber(chassisNumber) {
14. console.log('you are not allowed to change the chassis number');
15. }
17. // Methods
18. drive() {
19. console.log(`${this.brand} ${this.color} is driving`);
20. }
22. reverse() {
23. console.log(`${this.brand} ${this.color} is reversing`);
24. }
26. turn(direction) {
27. console.log(`${this.brand} ${this.color} is turning ${direction}`);
28. }
30. \_generateChassisNumber() {
31. return `${this.brand}-${Math.floor(Math.random() \* 1000)}`;
32. }
33. }

### Member Visibility

Member visibility bisa disebut juga sebagai hak akses pada sebuah properti dan method di dalam class. Secara default, seluruh properti dan method yang dibuat di dalam class bersifat public, alias dapat diakses di luar dari kode class via instance. Selain public, kita juga bisa membuat properti dan method bersifat private, terutama ketika kita ingin properti atau method tersebut hanya digunakan dalam cakupan kode di dalam class saja (penggunaan internal).

Di pembahasan sebelumnya, kita sudah mengenal bahwa pemberian tanda underscore pada properti atau method bisa dijadikan sebagai penanda bahwa ia dianggap private. Masalahnya adalah cara tersebut tidak benar-benar memproteksi hak aksesnya. Contoh, pada class Car, kita sudah membuat properti \_chassisNumber dan method \_generateChassisNumber() dengan tanda underscore. Namun, kedua member tersebut tetap saja masih bisa di akses via instance, bahkan kita bisa mengubah nilainya.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/15895?from=13933#run6-editor1)



1

class Car {

2

constructor(brand, color, maxSpeed) {

3

this.brand = brand;

4

this.color = color;

5

this.maxSpeed = maxSpeed;

6

this.\_chassisNumber = this.\_generateChassisNumber();

7

}

8

9

get chassisNumber() {

10

return this.\_chassisNumber;

11

}

12

13

set chassisNumber(chassisNumber) {

14

console.log('you are not allowed to change the chassis number');

15

}

16

17

// Methods

18

drive() {

19

console.log(`${this.brand} ${this.color} is driving`);

20

}

 Input  Reset

 Jalankan

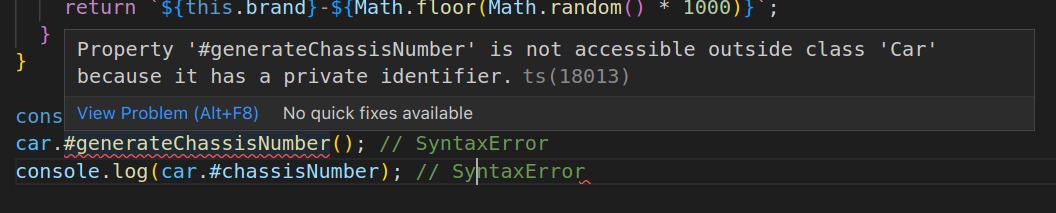
Untuk menyelesaikan masalah ini, JavaScript versi ES2022 secara resmi mengenalkan cara dalam menetapkan hak akses private pada properti dan method objek, yakni dengan menambahkan tanda # di awal penamaan properti atau method.

1. class Car {
2. #chassisNumber = null;
4. constructor(brand, color, maxSpeed) {
5. this.brand = brand;
6. this.color = color;
7. this.maxSpeed = maxSpeed;
8. this.#chassisNumber = this.#generateChassisNumber();
9. }
11. get chassisNumber() {
12. return this.#chassisNumber;
13. }
15. set chassisNumber(chassisNumber) {
16. console.log('you are not allowed to change the chassis number');
17. }
19. // Methods
20. drive() {
21. console.log(`${this.brand} ${this.color} is driving`);
22. }
24. reverse() {
25. console.log(`${this.brand} ${this.color} is reversing`);
26. }
28. turn(direction) {
29. console.log(`${this.brand} ${this.color} is turning ${direction}`);
30. }
32. #generateChassisNumber() {
33. return `${this.brand}-${Math.floor(Math.random() \* 1000)}`;
34. }
35. }

Khusus untuk properti yang sifatnya private, Anda harus mendeklarasikan propertinya di enclosing class seperti ini.

1. class Car {
2. #chassisNumber = null; // enclosing class
4. constructor(brand, color, maxSpeed) {
5. this.brand = brand;
6. this.color = color;
7. this.maxSpeed = maxSpeed;
8. this.#chassisNumber = this.#generateChassisNumber();
9. }
11. // ... kode lainnya.
12. }

Dengan begitu, properti dan method yang sifatnya private tidak dapat diakses di luar dari cakupan kode class, misalnya via instance.



## Pewarisan

Dalam gambaran dunia nyata, banyak sekali objek yang berbeda, tetapi punya kemiripan tertentu. Jika kita berbicara mobil, tentu banyak ragam dari mobil mulai dari mobil transportasi, mobil balap, ambulan, truk, dan mobil yang lainnya. Walaupun semua ragam tersebut termasuk dalam kategori mobil, tetapi kemampuannya berbeda-beda. Contoh, mobil balap memiliki kemampuan untuk mengaktifkan mode sport, sehingga dapat melaju dengan cepat; mobil ambulan dapat mengaktifkan sinyal darurat; mobil truk dapat menggerakan container-nya untuk menurunkan muatan. Di sisi lain, mereka memiliki kesamaan yaitu sama-sama mobil yang memiliki merek, warna, kecepatan maksimal, dan nomor rangka.

Sama halnya pada objek pada pemrograman, kita sering sekali mendapati kasus membuat sebuah objek dengan spesifikasi yang serupa, tetapi memiliki beberapa perbedaan kecil. Contoh, objek EmailService dengan WhatsAppService. Kedua objek tersebut sama-sama layanan perpesanan, mereka dapat mengirim pesan dan membutuhkan properti sender. Namun, terdapat beberapa perbedaan contohnya WhatsApp bisa mengirim pesan secara broadcast, sedangkan email bisa mengirim pesan secara delay.

Pertanyaannya, bagaimana cara membuat class untuk kedua objek ini? Sebenarnya kita bisa saja membuat dua class, yakni WhatsAppService dan EmailService dengan cara seperti ini.

1. class WhatsAppService {
2. constructor(sender) {
3. this.sender = sender;
4. }
6. sendMessage(message, receiver) {
7. console.log(`${this.sender} sent ${message} to ${receiver}`);
8. }
10. sendBroadcastMessage(message, receivers) {
11. for (const receiver of receivers) {
12. this.sendMessage(message, receiver);
13. }
14. }
15. }
17. class EmailService {
18. constructor(sender) {
19. this.sender = sender;
20. }
22. sendMessage(message, receiver) {
23. console.log(`${this.sender} sent ${message} to ${receiver}`);
24. }
26. sendDelayedMessage(message, receiver, delay) {
27. setTimeout(() => {
28. this.sendMessage(message, receiver);
29. }, delay);
30. }
31. }

Namun, jika kita lihat kode di atas, terdapat duplikasi kode untuk bagian yang “sama” antarkedua objek tersebut. Walau terlihat sederhana, tetapi tidak menutup kemungkinan kesamaan antarobjek tersebut terus berkembang dan kita perlu melakukan duplikasi kode di antara keduanya.

Oke, sekarang masalahnya adalah duplikasi kode, bagaimana cara menghindari duplikasi kode pada kasus ini? Sebenarnya, kita bisa saja membuat satu class yang mencakup kemampuan kedua objek tersebut. Sehingga kita bisa membuat instance WhatsAppService dan EmailService menggunakan satu class saja.

1. class MailService {
2. constructor(sender) {
3. this.sender = sender;
4. }
6. sendMessage(message, receiver) {
7. console.log(`${this.sender} sent ${message} to ${receiver}`);
8. }
10. sendDelayedMessage(message, receiver, delay) {
11. setTimeout(() => {
12. this.sendMessage(message, receiver);
13. }, delay);
14. }
16. sendBroadcastMessage(message, receivers) {
17. for (const receiver of receivers) {
18. this.sendMessage(message, receiver);
19. }
20. }
21. }
23. const whatsapp = new MailService('+6281234567890');
24. const email = new MailService('dimas@dicoding.com');

Namun, cara ini membuat objek whatsapp dan email tidak memiliki perbedaan. Masalah yang ditimbulkan adalah terdapat kemampuan yang tidak dibutuhkan di antara kedua objek tersebut, seperti sendDelayedMessage() di whatsApp dan sendBroadcastMessage() yang di email.

1. const whatsapp = new MailService('+6281234567890');
2. const email = new MailService('dimas@dicoding.com');
4. whatsapp.sendDelayedMessage(); // ???
5. email.sendBroadcastMessage(); // ???

Paradigma OOP menawarkan solusi dalam memecahkan masalah ini dengan konsep pewarisan atau lebih dikenal dengan istilah inheritance.

Dengan konsep inheritance, kita bisa mewariskan sifat-sifat yang berada di dalam sebuah class ke class lain. Konsep inheritance cocok ketika kita ingin membuat objek yang mirip dan memiliki sedikit perbedaan seperti kasus yang kita hadapi.

Implementasinya, kita tampung seluruh sifat yang “sama” pada sebuah class induk (superclass) dan sifat tersebut nantinya diwarisi kepada class di bawahnya (subclass). Kemudian pada subclass, kita bisa menambahkan kemampuan lain yang lebih spesifik.

Contohnya, kita buat superclass bernama MailService yang mengandung seluruh sifat yang sama pada WhatsAppService dan EmailService.

1. // Superclass
2. class MailService {
3. constructor(sender) {
4. this.sender = sender;
5. }
7. sendMessage(message, receiver) {
8. console.log(`${this.sender} sent ${message} to ${receiver}`);
9. }
10. }

Kemudian kita warisi sifat dari MailService ke subclass WhatsAppService dan EmailService dengan menggunakan keyword extends seperti ini.

1. // Superclass
2. class MailService {
3. constructor(sender) {
4. this.sender = sender;
5. }
7. sendMessage(message, receiver) {
8. console.log(`${this.sender} sent ${message} to ${receiver}`);
9. }
10. }
12. // Subclass
13. class WhatsAppService extends MailService {
15. }
17. // Subclass
18. class EmailService extends MailService {
20. }

Di dalam masing-masing subclass, kita bisa mendefinisikan method yang spesifik, seperti sendBroadcastMessage() untuk WhatsAppService dan sendDelayedMessage() untuk EmailService.

1. // Subclass
2. class WhatsAppService extends MailService {
3. sendBroadcastMessage(message, receivers) {
4. for (const receiver of receivers) {
5. this.sendMessage(message, receiver);
6. }
7. }
8. }
10. // Subclass
11. class EmailService extends MailService {
12. sendDelayedMessage(message, receiver, delay) {
13. setTimeout(() => {
14. this.sendMessage(message, receiver);
15. }, delay);
16. }
17. }

Lihatlah bahwa di dalam subclass WhatsAppService dan EmailService kita tetap memiliki akses terhadap method dari superclass melalui keyword this karena subclass mewarisi sifat dari superclass.

Dengan teknik pewarisan seperti ini, akhirnya kita bisa membuat dua objek serupa dengan cara yang jauh lebih efektif.

1. const whatsapp = new WhatsAppService('+6281234567890');
2. const email = new EmailService('dimas@dicoding.com');
4. whatsapp.sendMessage('Hello', '+6289876543210');
5. whatsapp.sendBroadcastMessage('Hello', ['+6289876543210', '+6282234567890']);
6. whatsapp.sendDelayedMessage(); // Error
8. email.sendMessage('Hello', 'john@doe.com');
9. email.sendDelayedMessage('Hello', 'john@doe.com', 3000);
10. email.sendBroadcastMessage(); // Error

### Pewarisan tanpa ES6 Class

Di awal penjelasan modul OOP, kami menunjukkan dua cara dalam membuat class atau constructor function. Anda sudah mengetahui cara melakukan pewarisan dengan sintaks class, tetapi bagaimana ya jika pewarisan tersebut dilakukan tanpa atau sebelum hadirnya sintaks class ES6?

Caranya adalah dengan teknik prototype inheritance seperti ini.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13937?from=15895#run1-editor1)



1

function MailService(sender) {

2

this.sender = sender;

3

}

4

5

MailService.prototype.sendMessage = function (message, receiver) {

6

console.log(`${this.sender} sent ${message} to ${receiver}`);

7

}

8

9

function WhatsAppService(sender) {

10

MailService.call(this, sender);

11

}

12

13

// Prototype inheritance

14

WhatsAppService.prototype = Object.create(MailService.prototype);

15

WhatsAppService.prototype.constructor = WhatsAppService;

16

17

WhatsAppService.prototype.sendBroadcastMessage = function (message, receivers) {

18

for (const receiver of receivers) {

19

this.sendMessage(message, receiver);

20

}

 Input  Reset

 Jalankan

Pewarisan dengan menggunakan keyword function memang lebih sulit untuk dibaca, alasan inilah yang membuat sintaks class hadir pada ES6. Namun, meskipun sintaks class terkesan sangat jauh berbeda dari function, sebenarnya implementasi keduanya sama-sama menggunakan prototype inheritance. Ingat! Class hanyalah cara lain dalam membuat constructor function.

### Operator instanceof

Ketika menulis kode, kita seringkali kita perlu mengecek jenis dari objek tersebut. Salah satu cara mengetahui jenis objek adalah dengan mengecek rantai prototype-nya. Nah, untuk mengetes sebuah objek berdasarkan prototype dari constructor function atau class tertentu, kita bisa menggunakan operator instanceof.

1. operand1 instanceof operand2

Penjelasannya:

* operand1: merupakan objek yang ingin dites prototype-nya.
* operand2: merupakan constructor function atau class.

Berikut contoh penggunaan dari operator instanceof dalam mengecek objek whatsapp yang merupakan instance dari class WhatsAppService.

1. const whatsapp = new WhatsAppService('+6281234567890');
3. console.log(whatsapp instanceof WhatsAppService); // true
4. console.log(whatsapp instanceof EmailService); // false

Operator instanceof mengembalikan boolean. Operasinya akan menghasilkan nilai true jika objek yang dites (operand pertama) memiliki prototype yang merupakan operand kedua. Jika prototype operand pertama bukanlah operand kedua, operasinya akan menghasilkan nilai false.

Operator instanceof juga akan mengecek prototype secara berantai. Artinya, instanceof juga mengecek hingga prototype yang diwarisi oleh objek tersebut.

1. const whatsapp = new WhatsAppService('+6281234567890');
2. const email = new EmailService('dimas@dicoding.com');
4. console.log(whatsapp instanceof WhatsAppService); // true
5. console.log(whatsapp instanceof EmailService); // false
6. console.log(whatsapp instanceof MailService); // true
8. console.log(email instanceof EmailService); // true
9. console.log(email instanceof WhatsAppService); // false
10. console.log(email instanceof MailService); // true

## Overriding

Konsep lain yang lekat dengan pewarisan adalah overriding. Overriding di dalam OOP merupakan fitur yang memperbolehkan subclass mendefinisikan implementasinya sendiri pada sebuah method yang sebenarnya sudah didefinisikan pada superclass-nya. Overriding diterapkan ketika kita ingin menetapkan implementasi yang spesifik di subclass pada sebuah method yang berasal dari superclass. Overriding juga biasa diterapkan ketika kita ingin menambah properti baru secara spesifik pada sebuah subclass.

Di JavaScript dan bahasa pemrograman yang menerapkan paradigma OOP lainnya, overriding umumnya dapat dilakukan pada constructor dan method class.

### Constuctor Overriding

Constructor merupakan method spesial pada class yang akan dipanggil pada saat pembuatan instance, lebih tepatnya ketika menggunakan keyword new diikuti dengan nama class.

1. const mailService = new MailService(); // akan memanggil method constructor pada class MailService

Constructor akan mengembalikan objek yang merupakan instance dari class tersebut. Perbedaan dengan method pada umumnya, pada constructor kita tidak perlu menulis return ketika hendak mengembalikan nilai karena sudah dilakukan secara implisit.

Di dalam constructor, biasanya kita mendefinisikan properti dan menetapkan nilainya. Contoh, pada superclass MailService, kita mendefinisikan properti sender yang nilainya diambil dari argumen method constructor.

1. class MailService {
2. constructor(sender) {
3. this.sender = sender;
4. }
6. // ... method lain disembunyikan
7. }
9. const mailService = new MailService('someReceiver');
10. console.log(mailService);
12. /\* Output:
13. MailService { sender: 'someReceiver' }
14. \*/

Ketika kita melakukan pewarisan, sering kali kita perlu menambahkan properti baru yang spesifik hanya pada subclass tertentu. Contoh, pada WhatsAppService yang merupakan subclass dari MailService, kita membutuhkan properti yang mengindikasikan akun bisnis atau bukan, sehingga kita perlu membuat properti baru bernama isBusiness hanya pada subclass tersebut. Agar bisa melakukan ini, kita perlu melakukan constructor overriding.

1. class MailService {
2. constructor(sender) {
3. this.sender = sender;
4. }
5. }
7. class WhatsAppService extends MailService {
8. // overriding constructor
9. constructor(sender, isBusiness) {
10. super(sender);
12. this.isBusiness = isBusiness;
13. }
14. }

Overriding constructor dilakukan sesimpel kita mendefinisikan kembali method constructor() pada sebuah subclass. Selain itu, method constructor didefinisikan dengan perubahan sesuai kebutuhan kita, misalnya menambahkan menambahkan properti dan argumen isBusiness. Satu hal yang penting dalam penerapan constructor overriding adalah jangan sampai lupa untuk memanggil method super().

Memanggil method super() artinya memanggil constructor superclass, yaitu MailService. Dengan begitu, kita tidak perlu mendefinisikan dan menetapkan nilai properti umum satu per satu. Meskipun dalam contoh kali ini properti yang bersifat umum hanya ada satu, yaitu sender, tetapi jika jumlah propertinya bertambah, kita tidak perlu menulis kode yang berulang pada setiap subclass.

Pada sintaks class, pemanggilan super() tidak boleh terlewatkan karena akan menyebabkan [ReferenceError](https://developer.mozilla.org/id/docs/Web/JavaScript/Reference/Global_Objects/ReferenceError).

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13938?from=13937#run1-editor1)



1

class MailService {

2

constructor(sender) {

3

this.sender = sender;

4

}

5

}

6

7

class WhatsAppService extends MailService {

8

constructor(sender, isBusiness) {

9

this.sender = sender;

10

this.isBusiness = isBusiness;

11

}

12

}

13

14

const whatsapp = new WhatsAppService('+6281234567890', true);

15

16

/\*\*

17

\* Output:

18

\* ReferenceError: Must call super constructor in derived class before accessing 'this' or returning from derived constructor

19

\*/

 Input  Reset

 Jalankan

### Method Overriding

Konsep overriding juga bisa diterapkan pada method class. Method overriding biasanya dilakukan ketika kita ingin mengubah implementasi method warisan superclass. Contohnya, pada class MailService terdapat method send() yang sudah didefinisikan implementasinya. Namun, jika pada subclass WhatsAppService kita membutuhkan implementasi yang berbeda, kita bisa override method send().

Cara override method pada subclass mirip seperti override constructor, tetapi kita tidak perlu memanggil method super di dalamnya.

1. class MailService {
2. constructor(sender) {
3. this.sender = sender;
4. }
6. sendMessage(message, receiver) {
7. console.log(`${this.sender} sent ${message} to ${receiver}`);
8. }
9. }
11. class WhatsAppService extends MailService {
12. constructor(sender, isBusiness) {
13. super(sender);
14. this.isBusiness = isBusiness;
15. }
17. // Overriding method
18. sendMessage(message, receiver) {
19. console.log(`${this.sender} sent ${message} to ${receiver} via WhatsApp`);
20. }
21. }

24. const mailService = new MailService('someSender');
25. const whatsappService = new WhatsAppService('+6281234567890', true);
27. mailService.sendMessage('Hai, apa kabar?', 'someReceiver');
28. whatsappService.sendMessage('Hai, apa kabar?', '+6289876543210');

31. /\*\*
32. \* Output:
33. \* someSender sent Hai, apa kabar? to someReceiver
34. \* +6281234567890 sent Hai, apa kabar? to +6289876543210 via WhatsApp
35. \*/

Jika dibutuhkan, kita juga bisa memanggil method sendMessage() dari superclass melalui keyword super.sendMessage().

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13938?from=13937#run2-editor1)



1

class MailService {

2

constructor(sender) {

3

this.sender = sender;

4

}

5

6

sendMessage(message, receiver) {

7

console.log(`${this.sender} sent ${message} to ${receiver}`);

8

}

9

​

10

}

11

​

12

class WhatsAppService extends MailService {

13

constructor(sender, isBusiness) {

14

super(sender);

15

this.isBusiness = isBusiness;

16

}

17

18

19

// Overriding method

20

sendMessage(message, receiver) {

 Input  Reset

 Jalankan

## Object Composition

Fitur pewarisan pada class begitu legit dalam memangkas duplikasi kode. Kita tidak perlu repetitif menulis kode yang sama ketika dua class atau lebih memiliki kemampuan yang sama. Namun, apakah pewarisan di JavaScript mampu memecahkan masalah yang lebih kompleks? Mari kita cari tahu.

Di dunia nyata, ada banyak peran dari seorang Developer, tetapi ada empat peran yang mungkin Anda sudah familier seperti Front-End Developer, Back-End Developer, DevOps, dan Full-Stack Developer. Semuanya masuk ke dalam kategori Developer dan mereka mampu menulis dan mengubah kode (commit changes). Selain itu, terdapat juga kemampuan yang spesifik terhadap perannya masing-masing, Front-End Developer dapat membangun antarmuka (build UI), Back-End Developer dapat membangun API (build API), dan DevOps mampu men-deploy aplikasi. Peran Full-Stack Developer adalah satu-satunya peran yang mampu melakukan banyak hal, di samping menulis dan mengubah kode, dia mampu membagun antarmuka, membangun API, dan men-deploy aplikasi.

Jika kita gambarkan skenario di atas dalam konsep OOP, Developer merupakan sebuah superclass; sedangkan Front-End Developer, Back-End Developer, dan DevOps adalah subclass. Struktur dari pewarisan class akan tampak seperti kode di bawah ini.

1. class Developer {
2. constructor(name) {
3. this.name = name;
4. }
6. commitChanges() {
7. console.log(`${this.name} is committing changes...`);
8. }
9. }
11. class FrontEndDeveloper extends Developer {
12. buildUI() {
13. console.log(`${this.name} is building UI...`);
14. }
15. }
17. class BackEndDeveloper extends Developer {
18. buildAPI() {
19. console.log(`${this.name} is building API...`);
20. }
21. }
23. class DevOps extends Developer {
24. deployApp() {
25. console.log(`${this.name} is deploying app...`);
26. }
27. }

Seluruh class sudah terdefinisikan dengan baik, tetapi kurang Full-Stack Developer. Apakah Anda punya ide cara mendefinisikan class FullStackDeveloper yang efektif tanpa ada kode repetitif menggunakan pewarisan?

1. class FullStackDeveloper extends Developer {
2. // ???
3. }

Jika menggunakan pewarisan tentu tidak ada cara selain kita menulis kembali method buildUI(), buildAPI(), dan deployApp() di dalam FullStackDeveloper.

1. class FullStackDeveloper extends Developer {
2. buildUI() {
3. console.log(`${this.name} is building UI...`);
4. }
6. buildAPI() {
7. console.log(`${this.name} is building API...`);
8. }
10. deployApp() {
11. console.log(`${this.name} is deploying app...`);
12. }
13. }

Cara ini sungguh tidak efektif karena jika terjadi perubahan pada salah satu fungsi, kita perlu mengubahnya di dua tempat.

Lalu apa solusinya?

Solusinya adalah meninggalkan pewarisan dan gunakan teknik object composition. Jika Anda lihat kembali kode di atas, pewarisan mendorong kita untuk menstrukturkan kode dengan pendekatan **peran atau identitas class**, yakni Front-End Developer, Back-End Developer, DevOps, dan Full-Stack Developer. Sedangkan object composition, dia tidak memperdulikan peran, melainkan kode distrukturkan berdasarkan **kemampuan yang dapat dilakukan**, seperti buildUI(), buildAPI(), dan deployApp().

1. class Developer {
2. constructor(name) {
3. this.name = name;
4. }
6. commitChanges() {
7. console.log(`${this.name} is committing changes...`);
8. }
9. }
11. function canBuildUI(developer) {
12. return {
13. buildUI: () => {
14. console.log(`${developer.name} is building UI...`);
15. }
16. }
17. }
19. function canBuildAPI(developer) {
20. return {
21. buildAPI: () => {
22. console.log(`${developer.name} is building API...`);
23. }
24. }
25. }
27. function canDeployApp(developer) {
28. return {
29. deployApp: () => {
30. console.log(`${developer.name} is deploying app...`);
31. }
32. }
33. }

Dengan memecah kode berdasarkan kemampuan, ke depannya kita akan lebih mudah dalam membuat objek dengan peran apa pun, mau Front-End Developer, Back-End Developer, DevOps, Full-Stack, atau peran lain yang membutuhkan kombinasi dari kemampuan-kemampuan yang ada, tanpa ada kode yang repetitif.

Agar lebih mudah dalam membuat objek, kita bisa membuat sebuah fungsi sebagai object creator dengan mengomposisikan kemampuan-kemampuan yang dibutuhkan. Di JavaScript, kita bisa secara mudah mengomposisikan objek dengan menggunakan method Object.assign().

1. function createFrontEndDeveloper(name) {
2. const developer = new Developer(name);
3. return Object.assign(developer, canBuildUI(developer));
4. }
6. function createBackEndDeveloper(name) {
7. const developer = new Developer(name);
8. return Object.assign(developer, canBuildAPI(developer));
9. }
11. function createDevOps(name) {
12. const developer = new Developer(name);
13. return Object.assign(developer, canDeployApp(developer));
14. }
16. function createFullStackDeveloper(name) {
17. const developer = new Developer(name);
18. return Object.assign(developer, canBuildUI(developer), canBuildAPI(developer), canDeployApp(developer));
19. }

Setelah membuat fungsi object creator, kita bisa secara mudah membuat objek Front-End Developer, Back-End Developer, DevOps, dan Full-Stack Developer.

Berikut implementasi utuh dari contoh kode di atas.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13942?from=13938#run1-editor1)



1

class Developer {

2

constructor(name) {

3

this.name = name;

4

}

5

6

commitChanges() {

7

console.log(`${this.name} is committing changes...`);

8

}

9

}

10

11

function canBuildUI(developer) {

12

return {

13

buildUI: () => {

14

console.log(`${developer.name} is building UI...`);

15

}

16

}

17

}

18

19

function canBuildAPI(developer) {

20

return {

 Input  Reset

 Jalankan

## Built-In Class

Di JavaScript sudah terdapat built-in class atau class bawaan, seperti Date, Object, Array, Math, dan String. Class bawaan dapat kita manfaatkan untuk berbagai hal, mulai dari manipulasi data-data terkait dengan array, operasi matematika, manipulasi karakter, ataupun manipulasi objek.

Mari kita ambil contoh class Date.

Date merupakan class bawaan JavaScript yang digunakan untuk utilitas terkait tanggal dan waktu. Class ini sangat membantu ketika program yang kita buat terdapat penggunaan dan manipulasi tanggal dan waktu. Kita bisa mudah untuk mendapatkan waktu dalam bentuk instance Date atau mengonversinya ke dalam berbagai format yang kita inginkan.

Berikut contoh dalam penggunaan class bawaan Date.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13947?from=13942#run1-editor1)



1

const date = new Date();

2

3

const timeInJakarta = date.toLocaleString('id-ID', {

4

timeZone: 'Asia/Jakarta',

5

});

6

7

const timeInTokyo = date.toLocaleString('ja-JP', {

8

timeZone: 'Asia/Tokyo',

9

});

10

11

const timeInMakassar = date.toLocaleString('id-ID', {

12

timeZone: 'Asia/Makassar',

13

});

14

15

console.log(timeInJakarta);

16

console.log(timeInTokyo);

17

console.log(timeInMakassar);

18

19

/\*\*

20

\* Output:

 Input  Reset

 Jalankan

Lihat! Betapa mudahnya kita mendapatkan data waktu di JavaScript dengan menggunakan class bawaan Date. Selain yang kami contohkan, masih banyak hal yang bisa kita lakukan dengan class Date. Anda bisa eksplorasi fitur-fitur dari class Date pada dokumentasi berikut: [Date() constructor](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/Date#:~:text=See%20also-,date()%20constructor,-The%20Date()%20constructor).

Mari kita berbicara tentang class bawaan lain, yang menjadi salah satu favorite kami adalah Array.

Dengan class Array, kita bisa membuat struktur data dalam bentuk array.

1. const myArray = new Array('a', 'b', 'c', 'a', 'b', 'c');
2. console.log(myArray); // ['a', 'b', 'c', 'a', 'b', 'c']

Sekilas, tidak ada spesial dari class ini karena sebenarnya kita bisa membuat array seperti itu dengan menggunakan array literals (tanda []). Namun, dengan class Array, kita bisa menerapkan konsep pewarisan untuk membuat subclass baru yang dapat kita modifikasi sesuai kebutuhan.

Contohnya, kita ingin membuat sebuah struktur data mirip array, tetapi nilai tiap elemennya unik, maka kita bisa membuat subclass UniqueArray yang mewarisi class Array. Di dalam subclass UniqueArray, implementasinya tampak seperti ini.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13947?from=13942#run2-editor1)



1

class UniqueArray extends Array {

2

constructor(...args) {

3

// make sure args is unique before passing it to super

4

const uniqueValue = args.filter((item, index) => args.indexOf(item) === index);

5

6

super(...uniqueValue);

7

}

8

9

push(item) {

10

// make sure only unique item is added

11

if (!this.includes(item)) {

12

super.push(item);

13

}

14

}

15

}

16

17

const someArray = new UniqueArray('a', 'b', 'c', 'a', 'b', 'c');

18

console.log(someArray); // ['a', 'b', 'c']

19

someArray.push('d');

20

console.log(someArray); // ['a', 'b', 'c', 'd']

 Input  Reset

 Jalankan

Menarik, bukan? Dengan konsep OOP, kita bisa extends kemampuan JavaScript lebih dari standar yang ada.

Kami tekankan lagi, di samping Date dan Array, terdapat banyak class bawaan JavaScript yang bisa Anda eksplorasi. Pemahaman Anda terhadap konsep-konsep OOP bisa membuka jalan untuk menambahkan atau memodifikasi kemampuan standar JavaScript.

Berikut beberapa class bawaan JavaScript yang bisa Anda eksplorasi secara mandiri:

* [Date](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/Date)
* [Object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/Object)
* [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/Array)
* [Math](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math)
* [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)
* [Number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number/Number)

## Kuis Coding: Object-Oriented Programming

Untuk menguji kemampuan praktikal Anda dalam memahami materi Object Oriented Programming di JavaScript, silakan selesaikan kuis berikut.

Sebelum Anda mengerjakan kuis, ada beberapa hal yang perlu Anda perhatikan.

* Mohon untuk membaca secara seksama perintah, kriteria, atau soal pada komentar kode yang diberi tanda **TODO.**
* Lakukan pengujian pada kode yang Anda tulis contohnya dengan menggunakan console.log() untuk memastikan hasil sesuai dengan yang Anda harapkan. Gunakan tombol **Jalankan**.

Bila sudah yakin dengan pekerjaan yang Anda lakukan, silakan submit kode dengan klik tombol **Submit kode**. Jika kode yang di-submit benar, Anda bisa melanjutkan ke modul berikutnya. Jika tidak, Anda bisa memperbaiki kembali kode sebanyak yang Anda mau.

Batas waktu tunggu untuk mengirimkan kembali perbaikan kode adalah 1 menit. Jadi pastikan kode yang Anda tulis sudah memenuhi perintah, kriteria, atau soal pada kuis kali ini.

Good luck!

|  |
| --- |
| **Mengalami kendala?** Cobalah kunjungi [Forum Diskusi](https://www.dicoding.com/academies/256/discussions?tutorial=15910) untuk mencari solusi atau bertanya langsung kepada mentor kami. |

Selamat, Anda telah menyelesaikan modul ini.  
Kode yang Anda kirimkan telah berfungsi dengan baik.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/15910?from=13947#run1-editor1)



67

​

68

super(name, age, false);

69

​

70

}

71

​

72

fly() {

73

​

74

return `${this.name} sedang terbang!`;

75

​

76

};

77

​

78

}

79

​

80

const myRabbit = new Rabbit('Labi',2);

81

​

82

const myEagle = new Eagle('Elo',4);

83

​

84

myRabbit.eat()

85

​

86

myEagle.fly()

87

​

88

console.log(myRabbit)

89

​

90

console.log(myRabbit.eat());

91

​

92

console.log(myEagle)

93

​

94

console.log(myEagle.fly());

95

​

96

​

97

/\*\*

98

\* Jangan hapus kode di bawah ini

99

\*/

100

​

101

module.exports = {

102

Animal, Rabbit, Eagle, myRabbit, myEagle,

103

};

## Rangkuman Materi

### Object-Oriented Programming

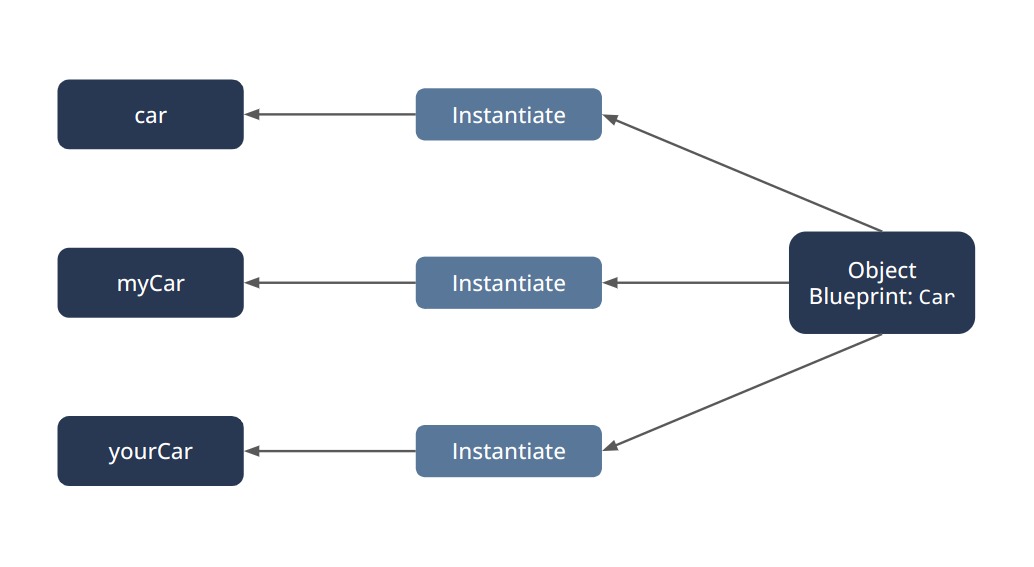
Object-Oriented Programming (OOP) adalah salah satu paradigma dalam pemrograman yang berfokus pada pembuatan sebuah objek dan interaksi dengan objek-objek tersebut. Di OOP, objek merupakan sebuah entitas yang terdiri dari dua hal, yakni properties dan methods.

* Properti merupakan nilai di dalam objek yang menyimpan informasi tentang objek tersebut.
* Method merupakan fungsi yang menggambarkan aksi yang dapat dilakukan oleh objek tersebut.

Paradigma OOP kerap digambarkan dengan kehidupan dunia nyata. Objek di dalam OOP, bisa kita anggap seperti objek yang ada di dunia nyata. Kita ambil contoh, mobil. Dalam OOP, sebuah mobil dapat dilihat sebagai objek yang memiliki beberapa properti seperti merek, warna, kecepatan maksimal, dan nomor rangka. Objek tersebut memiliki kemampuan atau method seperti maju, mundur, dan belok.

Melalui object blueprint, kita bisa membuat cetakan untuk membuat objek yang sudah terdefinisikan macam-macam properti dan method-nya. Sehingga kita cukup menggunakan cetakan tersebut untuk membuat objek yang serupa, tetapi kita bisa menentukan nilai-nilai properti yang berbeda.

Contohnya, Anda bisa membuat sebuah blueprint bernama Car. Di dalam blueprint tersebut, Anda bisa definisikan properti-properti dan method yang ingin dimiliki objek mobil nantinya. Setelah membuat sebuah blueprint, Anda bisa dengan mudah membuat banyak objek mobil contohnya myCar, yourCar, dan dicodingCar dengan lebih mudah.

[](https://www.dicoding.com/academies/256/tutorials/15988?from=15910)

### Constructor Function

Di JavaScript, object blueprint dapat dibuat dengan menggunakan constructor function yang tampak seperti ini.

1. function Car(brand, color, maxSpeed, chassisNumber) {
2. this.brand = brand;
3. this.color = color;
4. this.maxSpeed = maxSpeed;
5. this.chassisNumber = chassisNumber;
6. }
8. Car.prototype.drive = function() {
9. console.log(`${this.brand} ${this.color} is driving`);
10. }
12. Car.prototype.reverse = function() {
13. console.log(`${this.brand} ${this.color} is reversing`);
14. }
16. Car.prototype.turn = function() {
17. console.log(`${this.brand} ${this.color} is turning`);
18. }

Kode di atas merupakan contoh membuat object blueprint dari sebuah mobil. Anda bisa membuat objek mobil--yang merupakan instance dari Car--dengan memanfaatkan constructor function Car.

1. function Car(brand, color, maxSpeed, chassisNumber) {
2. this.brand = brand;
3. this.color = color;
4. this.maxSpeed = maxSpeed;
5. this.chassisNumber = chassisNumber;
6. }
8. Car.prototype.drive = function() {
9. console.log(`${this.brand} ${this.color} is driving`);
10. }
12. Car.prototype.reverse = function() {
13. console.log(`${this.brand} ${this.color} is reversing`);
14. }
16. Car.prototype.turn = function() {
17. console.log(`${this.brand} ${this.color} is turning`);
18. }
20. // Membuat objek mobil dengan constructor function Car
21. const car1 = new Car('Toyota', 'Silver', 200, 'to-1');
22. const car2 = new Car('Honda', 'Black', 180, 'ho-1');
23. const car3 = new Car('Suzuki', 'Red', 220, 'su-1');
25. console.log(car1);
26. console.log(car2);
27. console.log(car3);
29. car1.drive();
30. car2.drive();
31. car3.drive();
33. /\* Output
34. Car { brand: 'Toyota', color: 'Silver', maxSpeed: 200, chassisNumber: 'to-1' }
35. Car { brand: 'Honda', color: 'Black', maxSpeed: 180, chassisNumber: 'ho-1' }
36. Car { brand: 'Suzuki', color: 'Red', maxSpeed: 220, chassisNumber: 'su-1' }
38. Toyota Silver is driving
39. Honda Black is driving
40. Suzuki Red is driving
41. \*/

#### Sintaks Class di ES6

Sintaks JavaScript modern (ES6) menawarkan cara membuat constructor function dengan menggunakan keyword class. Hal ini membuat penerapan OOP di JavaScript mirip seperti bahasa pemrograman berbasis class. Kita bisa menulis constructor function Car dengan gaya class seperti ini.

1. class Car {
2. constructor(brand, color, maxSpeed, chassisNumber) {
3. this.brand = brand;
4. this.color = color;
5. this.maxSpeed = maxSpeed;
6. this.chassisNumber = chassisNumber;
7. }
9. drive() {
10. console.log(`${this.brand} ${this.color} is driving`);
11. }
13. reverse() {
14. console.log(`${this.brand} ${this.color} is reversing`);
15. }
17. turn() {
18. console.log(`${this.brand} ${this.color} is turning`);
19. }
20. }
22. // Membuat objek mobil dengan constructor function Car
23. const car1 = new Car('Toyota', 'Silver', 200, 'to-1');
24. const car2 = new Car('Honda', 'Black', 180, 'ho-1');
25. const car3 = new Car('Suzuki', 'Red', 220, 'su-1');
27. console.log(car1);
28. console.log(car2);
29. console.log(car3);
31. car1.drive();
32. car2.drive();
33. car3.drive();
35. /\* Output
36. Car { brand: 'Toyota', color: 'Silver', maxSpeed: 200, chassisNumber: 'to-1' }
37. Car { brand: 'Honda', color: 'Black', maxSpeed: 180, chassisNumber: 'ho-1' }
38. Car { brand: 'Suzuki', color: 'Red', maxSpeed: 220, chassisNumber: 'su-1' }
40. Toyota Silver is driving
41. Honda Black is driving
42. Suzuki Red is driving
43. \*/

Jika Anda memiliki pengalaman di bahasa pemrograman C++, Java, atau C#, pembuatan class seperti di atas tentu lebih mudah dipahami, bukan? Memang inilah tujuan diciptakan sintaks class di JavaScript. Ia hadir untuk mempermudah penerapan OOP bagi developer yang memiliki latar belakang class-based language.

Walaupun di JavaScript sudah hadir sintaks class, hal itu tidak membuat JavaScript menjadi class-based language. Faktanya, sintaks class di JavaScript hanyalah syntactic sugar atau cara alternatif dalam mendefinisikan constructor function. Untuk membuktikan hal tersebut, kita bisa mengecek tipe class melalui operator typeof.

1. class Car {};
2. console.log(typeof Car);
4. /\* Output
5. function
6. \*/

### Properti dan Method

Di dalam sebuah class, kita dapat mendefinisikan dua hal, yaitu properti dan method.

#### Properti

Properti merupakan bagian dari class yang mendefinisikan nilai-nilai yang terkandung dalam sebuah instance class. Contohnya, jika Anda membuat class Car, properti adalah informasi yang sekiranya terdapat pada sebuah mobil seperti brand, color, maxSpeed, dan chasissNumber. Nilai dari properti biasanya diambil dari dari argumen constructor agar nilainya dapat bervariasi setiap kali membuat instance.

1. class Car {
2. constructor(brand, color, maxSpeed, chassisNumber) {
3. this.brand = brand;
4. this.color = color;
5. this.maxSpeed = maxSpeed;
6. this.chassisNumber = chassisNumber;
7. }
8. }
10. const car1 = new Car('BMW', 'red', 200, 'b-1');
11. const car2 = new Car('Audi', 'blue', 220, 'a-1');
12. const car3 = new Car('BMW', 'black', 250, 'b-2');
14. /\* Output:
15. Car { brand: 'BMW', color: 'red', maxSpeed: 200, chassisNumber: 'b-1' }
16. Car { brand: 'Audi', color: 'blue', maxSpeed: 220, chassisNumber: 'a-1' }
17. Car { brand: 'BMW', color: 'black', maxSpeed: 250, chassisNumber: 'b-2' }
18. \*/

Adakalanya nilai properti juga bisa didefinisikan di dalam class itu sendiri. Contohnya, kita tidak ingin pengguna menentukan nomor rangka mobil secara mandiri, maka kita bisa memberi nilai properti chassisNumber langsung di dalam fungsi constructor.

1. class Car {
2. constructor(brand, color, maxSpeed) {
3. this.brand = brand;
4. this.color = color;
5. this.maxSpeed = maxSpeed;
6. // Set a random chassis number
7. this.chassisNumber = `${brand}-${Math.floor(Math.random() \* 1000) + 1}`;
8. }
9. }
11. const car1 = new Car('BMW', 'red', 200);
12. const car2 = new Car('Audi', 'blue', 220);
13. const car3 = new Car('BMW', 'black', 250);
15. console.log(car1);
16. console.log(car2);
17. console.log(car3);
19. /\* Output:
20. Car { brand: 'BMW', color: 'red', maxSpeed: 200, chassisNumber: 'BMW-558' }
21. Car { brand: 'Audi', color: 'blue', maxSpeed: 220, chassisNumber: 'Audi-215' }
22. Car { brand: 'BMW', color: 'black', maxSpeed: 250, chassisNumber: 'BMW-799' }
23. \*/

##### Properti Getter dan Setter

Accessor property merupakan properti yang dikontrol oleh sebuah getter dan setter. Nilai yang didapatkan dari properti tersebut dikontrol oleh method get dan cara menetapkan nilai tersebut dikontrol oleh method set.

Catatan penting yang perlu Anda ketahui mengenai getter setter adalah:

1. method getter harus mengembalikan sebuah nilai dan nilai tersebut akan menjadi nilai properti;
2. method setter harus menerima satu argumen yang nilainya diambil dari operand ke dua ketika melakukan assignment operator.

Berikut contoh dari accessor property.

1. class User {
2. constructor(firstName, lastName) {
3. this.firstName = firstName;
4. this.lastName = lastName;
5. }
7. get fullName() {
8. return `${this.firstName} ${this.lastName}`;
9. }
11. set fullName(fullName) {
12. const [firstName, lastName] = fullName.split(' ');
13. this.firstName = firstName;
14. this.lastName = lastName;
15. }
16. }
18. const user = new User('John', 'Doe');
19. console.log(user);
20. console.log(user.fullName);
22. user.fullName = 'Fulan Fulanah';
23. console.log(user);
24. console.log(user.fullName);
26. /\* Output:
27. User { firstName: 'John', lastName: 'Doe' }
28. John Doe
29. User { firstName: 'Fulan', lastName: 'Fulanah' }
30. Fulan Fulanah
31. \*/

#### Method

Method adalah sebuah fungsi yang berada di dalam sebuah class dan dapat diakses melalui instance Class tersebut. Method biasanya mengindikasikan hal yang dapat dilakukan oleh sebuah class. Bila kita berbicara tentang class Car, method yang ada bisa drive(), reverse(), dan turn().

1. class Car {
2. constructor(brand, color, maxSpeed) {
3. this.brand = brand;
4. this.color = color;
5. this.maxSpeed = maxSpeed;
6. this.\_chassisNumber = `${brand}-${Math.floor(Math.random() \* 1000)}`;
7. }
9. get chassisNumber() {
10. return this.\_chassisNumber;
11. }
13. set chassisNumber(chassisNumber) {
14. console.log('you are not allowed to change the chassis number');
15. }
17. // Methods
18. drive() {
19. console.log(`${this.brand} ${this.color} is driving`);
20. }
22. reverse() {
23. console.log(`${this.brand} ${this.color} is reversing`);
24. }
26. turn(direction) {
27. console.log(`${this.brand} ${this.color} is turning ${direction}`);
28. }
29. }

Sama seperti fungsi JavaScript, method bisa menerima sebuah argumen. Contohnya pada method turn() di class Car, kami memanfaatkan argumen direction untuk menetapkan arah dalam menjalankan method-nya.

Selain argumen, method juga dapat memiliki akses ke nilai properti atau method lainnya melalui keyword this.

Method di dalam class hanya bisa dijalankan melalui instance dari class tersebut.

1. const car = new Car('BMW', 'red', 200);
3. car.drive();
4. car.turn('left');
5. car.reverse();
7. /\* Output:
8. BMW red is driving
9. BMW red is turning left
10. BMW red is reversing
11. \*/

#### Member Visibility

Member visibility bisa disebut juga sebagai hak akses pada sebuah properti dan method di dalam class. Secara default, seluruh properti dan method yang dibuat di dalam class bersifat public, alias dapat diakses di luar dari kode class via instance. Selain public, kita juga bisa membuat properti dan method bersifat private, terutama ketika kita ingin properti atau method tersebut hanya digunakan dalam cakupan kode di dalam class saja (penggunaan internal).

JavaScript versi ES2022 secara resmi mengenalkan cara dalam menetapkan hak akses private pada properti dan method objek, yakni dengan menambahkan tanda # di awal penamaan properti atau method.

1. class Car {
2. #chassisNumber = null;
4. constructor(brand, color, maxSpeed) {
5. this.brand = brand;
6. this.color = color;
7. this.maxSpeed = maxSpeed;
8. this.#chassisNumber = this.#generateChassisNumber();
9. }
11. get chassisNumber() {
12. return this.#chassisNumber;
13. }
15. set chassisNumber(chassisNumber) {
16. console.log('you are not allowed to change the chassis number');
17. }
19. // Methods
20. drive() {
21. console.log(`${this.brand} ${this.color} is driving`);
22. }
24. reverse() {
25. console.log(`${this.brand} ${this.color} is reversing`);
26. }
28. turn(direction) {
29. console.log(`${this.brand} ${this.color} is turning ${direction}`);
30. }
32. #generateChassisNumber() {
33. return `${this.brand}-${Math.floor(Math.random() \* 1000)}`;
34. }
35. }

### Pewarisan

Dalam pemrograman, kita sering sekali mendapati kasus membuat sebuah objek dengan spesifikasi yang serupa, tetapi memiliki beberapa perbedaan kecil. Contoh, objek EmailService dengan WhatsAppService. Kedua objek tersebut sama-sama layanan perpesanan, mereka dapat mengirim pesan dan membutuhkan properti sender. Namun, terdapat beberapa perbedaan contohnya WhatsApp bisa mengirim pesan secara broadcast, sedangkan email bisa mengirim pesan secara delay.

Paradigma OOP menawarkan solusi dalam memecahkan masalah ini dengan konsep pewarisan atau lebih dikenal dengan istilah inheritance.

Dengan konsep inheritance, kita bisa mewariskan sifat-sifat yang berada di dalam sebuah class ke class lain. Konsep inheritance cocok ketika kita ingin membuat objek yang mirip dan memiliki sedikit perbedaan seperti kasus yang kita hadapi.

Implementasinya, kita tampung seluruh sifat yang “sama” pada sebuah class induk (superclass) dan sifat tersebut nantinya diwarisi kepada class di bawahnya (subclass). Kemudian pada subclass, kita bisa menambahkan kemampuan lain yang lebih spesifik.

Contohnya, kita buat superclass bernama MailService yang mengandung seluruh sifat yang sama pada WhatsAppService dan EmailService.

1. // Superclass
2. class MailService {
3. constructor(sender) {
4. this.sender = sender;
5. }
7. sendMessage(message, receiver) {
8. console.log(`${this.sender} sent ${message} to ${receiver}`);
9. }
10. }

Kemudian kita warisi sifat dari MailService ke subclass WhatsAppService dan EmailService dengan menggunakan keyword extends seperti ini.

1. // Superclass
2. class MailService {
3. constructor(sender) {
4. this.sender = sender;
5. }
7. sendMessage(message, receiver) {
8. console.log(`${this.sender} sent ${message} to ${receiver}`);
9. }
10. }
12. // Subclass
13. class WhatsAppService extends MailService {
15. }
17. // Subclass
18. class EmailService extends MailService {
20. }

Di dalam masing-masing subclass, kita bisa mendefinisikan method yang spesifik, seperti sendBroadcastMessage() untuk WhatsAppService dan sendDelayedMessage() untuk EmailService.

1. // Subclass
2. class WhatsAppService extends MailService {
3. sendBroadcastMessage(message, receivers) {
4. for (const receiver of receivers) {
5. this.sendMessage(message, receiver);
6. }
7. }
8. }
10. // Subclass
11. class EmailService extends MailService {
12. sendDelayedMessage(message, receiver, delay) {
13. setTimeout(() => {
14. this.sendMessage(message, receiver);
15. }, delay);
16. }
17. }

Dengan teknik pewarisan seperti ini, akhirnya kita bisa membuat dua objek serupa dengan cara yang jauh lebih efektif.

1. const whatsapp = new WhatsAppService('+6281234567890');
2. const email = new EmailService('dimas@dicoding.com');
4. whatsapp.sendMessage('Hello', '+6289876543210');
5. whatsapp.sendBroadcastMessage('Hello', ['+6289876543210', '+6282234567890']);
6. whatsapp.sendDelayedMessage(); // Error
8. email.sendMessage('Hello', 'john@doe.com');
9. email.sendDelayedMessage('Hello', 'john@doe.com', 3000);
10. email.sendBroadcastMessage(); // Error

#### Operator instanceof

Untuk mengetes sebuah objek berdasarkan prototype dari constructor function atau class tertentu, kita bisa menggunakan operator instanceof.

1. operand1 instanceof operand2

Penjelasannya:

* operand1: merupakan objek yang ingin dites prototype-nya.
* operand2: merupakan constructor function atau class.

Berikut contoh penggunaan dari operator instanceof dalam mengecek objek whatsapp yang merupakan instance dari class WhatsAppService.

1. const whatsapp = new WhatsAppService('+6281234567890');
3. console.log(whatsapp instanceof WhatsAppService); // true
4. console.log(whatsapp instanceof EmailService); // false

Operator instanceof mengembalikan boolean. Operasinya akan menghasilkan nilai true jika objek yang dites (operand pertama) memiliki prototype yang merupakan operand kedua. Jika prototype operand pertama bukanlah operand kedua, operasinya akan menghasilkan nilai false.

Operator instanceof juga akan mengecek prototype secara berantai. Artinya, instanceof juga mengecek hingga prototype yang diwarisi oleh objek tersebut.

1. const whatsapp = new WhatsAppService('+6281234567890');
2. const email = new EmailService('dimas@dicoding.com');
4. console.log(whatsapp instanceof WhatsAppService); // true
5. console.log(whatsapp instanceof EmailService); // false
6. console.log(whatsapp instanceof MailService); // true
8. console.log(email instanceof EmailService); // true
9. console.log(email instanceof WhatsAppService); // false
10. console.log(email instanceof MailService); // true

### Overriding

Konsep lain yang lekat dengan pewarisan adalah overriding. Overriding di dalam OOP merupakan fitur yang memperbolehkan subclass mendefinisikan implementasinya sendiri pada sebuah method yang sebenarnya sudah didefinisikan pada superclass-nya. Overriding diterapkan ketika kita ingin menetapkan implementasi yang spesifik di subclass pada sebuah method yang berasal dari superclass. Overriding juga biasa diterapkan ketika kita ingin menambah properti baru secara spesifik pada sebuah subclass.

Di JavaScript dan bahasa pemrograman yang menerapkan paradigma OOP lainnya, overriding umumnya dapat dilakukan pada constructor dan method class.

#### Constuctor Overriding

Ketika kita melakukan pewarisan, sering kali kita perlu menambahkan properti baru yang spesifik hanya pada subclass tertentu. Contoh, pada WhatsAppService yang merupakan subclass dari MailService, kita membutuhkan properti yang mengindikasikan akun bisnis atau bukan, sehingga kita perlu membuat properti baru bernama isBusiness hanya pada subclass tersebut. Agar bisa melakukan ini, kita perlu melakukan constructor overriding.

1. class MailService {
2. constructor(sender) {
3. this.sender = sender;
4. }
5. }
7. class WhatsAppService extends MailService {
8. // overriding constructor
9. constructor(sender, isBusiness) {
10. super(sender);
12. this.isBusiness = isBusiness;
13. }
14. }

Overriding constructor dilakukan sesimpel kita mendefinisikan kembali method constructor() pada sebuah subclass. Selain itu, method constructor didefinisikan dengan perubahan sesuai kebutuhan kita, misalnya menambahkan menambahkan properti dan argumen isBusiness. Satu hal yang penting dalam penerapan constructor overriding adalah jangan sampai lupa untuk memanggil method super().

#### Method Overriding

Konsep overriding juga bisa diterapkan pada method class. Method overriding biasanya dilakukan ketika kita ingin mengubah implementasi method warisan superclass. Contohnya, pada class MailService terdapat method send() yang sudah didefinisikan implementasinya. Namun, jika pada subclass WhatsAppService kita membutuhkan implementasi yang berbeda, kita bisa override method send().

Cara override method pada subclass mirip seperti override constructor, tetapi kita tidak perlu memanggil method super di dalamnya. Jika dibutuhkan, kita juga bisa memanggil method sendMessage() dari superclass melalui keyword super.sendMessage().

1. class MailService {
2. constructor(sender) {
3. this.sender = sender;
4. }
6. sendMessage(message, receiver) {
7. console.log(`${this.sender} sent ${message} to ${receiver}`);
8. }
9. }
11. class WhatsAppService extends MailService {
12. constructor(sender, isBusiness) {
13. super(sender);
14. this.isBusiness = isBusiness;
15. }
17. // Overriding method
18. sendMessage(message, receiver) {
19. // memanggil method sendMessage pada superclass
20. super.sendMessage(message, receiver);
22. console.log('message sent via WhatsApp');
23. }
24. }

27. const mailService = new MailService('someSender');
28. const whatsappService = new WhatsAppService('+6281234567890', true);
30. mailService.sendMessage('Hai, apa kabar?', 'someReceiver');
31. whatsappService.sendMessage('Hai, apa kabar?', '+6289876543210');
33. /\*\*
34. \* Output:
35. \* someSender sent Hai, apa kabar? to
36. \* someSender sent Hai, apa kabar? to +6289876543210
37. \* message sent via WhatsApp
38. \*/

### Built-In Class

Di JavaScript sudah terdapat built-in class atau class bawaan, seperti Date, Object, Array, Math, dan String. Class bawaan dapat kita manfaatkan untuk berbagai hal, mulai dari manipulasi data-data terkait dengan array, operasi matematika, manipulasi karakter, ataupun manipulasi objek.

Mari kita ambil contoh class Date.

Date merupakan class bawaan JavaScript yang digunakan untuk utilitas terkait tanggal dan waktu. Class ini sangat membantu ketika program yang kita buat terdapat penggunaan dan manipulasi tanggal dan waktu. Kita bisa mudah untuk mendapatkan waktu dalam bentuk instance Date atau mengonversinya ke dalam berbagai format yang kita inginkan.

Berikut contoh dalam penggunaan class bawaan Date.

1. const date = new Date();
3. const timeInJakarta = date.toLocaleString('id-ID', {
4. timeZone: 'Asia/Jakarta',
5. });
7. const timeInTokyo = date.toLocaleString('ja-JP', {
8. timeZone: 'Asia/Tokyo',
9. });
11. const timeInMakassar = date.toLocaleString('id-ID', {
12. timeZone: 'Asia/Makassar',
13. });
15. console.log(timeInJakarta);
16. console.log(timeInTokyo);
17. console.log(timeInMakassar);
19. /\*\*
20. \* Output:
21. \* 22/12/2022 10.37.14
22. \* 2022/12/22 12:37:14
23. \* 22/12/2022 11.37.14
24. \*/

Lihat! Betapa mudahnya kita mendapatkan data waktu di JavaScript dengan menggunakan class bawaan Date. Selain yang kami contohkan, masih banyak hal yang bisa kita lakukan dengan class Date. Anda bisa eksplorasi fitur-fitur dari class Date pada dokumentasi berikut: [Date() constructor](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/Date#:~:text=See%20also-,date()%20constructor,-The%20Date()%20constructor).

Mari kita berbicara tentang class bawaan lain, yang menjadi salah satu favorite kami adalah Array.

Dengan class Array, kita bisa membuat struktur data dalam bentuk array.

1. const myArray = new Array('a', 'b', 'c', 'a', 'b', 'c');
2. console.log(myArray); // ['a', 'b', 'c', 'a', 'b', 'c']

Sekilas, tidak ada spesial dari class ini karena sebenarnya kita bisa membuat array seperti itu dengan menggunakan array literals (tanda []). Namun, dengan class Array, kita bisa menerapkan konsep pewarisan untuk membuat subclass baru yang dapat kita modifikasi sesuai kebutuhan.

Contohnya, kita ingin membuat sebuah struktur data mirip array, tetapi nilai tiap elemennya unik, maka kita bisa membuat subclass UniqueArray yang mewarisi class Array. Di dalam subclass UniqueArray, implementasinya tampak seperti ini.

1. class UniqueArray extends Array {
2. constructor(...args) {
3. // make sure args is unique before passing it to super
4. const uniqueValue = args.filter((item, index) => args.indexOf(item) === index);
6. super(...uniqueValue);
7. }
9. add(item) {
10. // make sure only unique item is added
11. if (!this.includes(item)) {
12. super.push(item);
13. }
14. }
15. }
17. const someArray = new UniqueArray('a', 'b', 'c', 'a', 'b', 'c');
18. console.log(someArray); // ['a', 'b', 'c']
19. someArray.add('d');
20. console.log(someArray); // ['a', 'b', 'c', 'd']
21. someArray.add('a');
22. console.log(someArray); // ['a', 'b', 'c', 'd']

Menarik, bukan? Dengan konsep OOP, kita bisa extends kemampuan JavaScript lebih dari standar yang ada.

Kami tekankan lagi, di samping Date dan Array, terdapat banyak class bawaan JavaScript yang bisa Anda eksplorasi. Pemahaman Anda terhadap konsep-konsep OOP bisa membuka jalan untuk menambahkan atau memodifikasi kemampuan standar JavaScript.

Berikut beberapa class bawaan JavaScript yang bisa Anda eksplorasi secara mandiri:

* [Date](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/Date)
* [Object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/Object)
* [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/Array)
* [Math](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math)
* [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)
* [Number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number/Number)

## Video Object Oriented Programming

Untuk memperdalam dan mempermudah pemahaman pada materi ini, Anda dapat menyimak video pembahasan berikut.

<https://youtu.be/Z5I46E3-QMU>

### Materi Pendukung

Berikut ini beberapa materi tambahan yang bisa Anda pelajari untuk mendalami seputar Object-Oriented Programming (OOP):

* [Object-oriented JavaScript for beginners](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object-oriented_JS)
* [A Guide to Object-Oriented Programming in JavaScript](https://betterprogramming.pub/object-oriented-programming-in-javascript-b3bda28d3e81)

Aturan

Knowledge Check : Object-Oriented Programming (OOP) bertujuan untuk menguji pengetahuan Anda tentang materi yang telah dipelajari di modul ini.

Terdapat 2 pertanyaan yang harus dikerjakan dalam kuis ini. Beberapa ketentuannya sebagai berikut:

* Syarat nilai kelulusan : 100%
* Durasi ujian : 5 menit

Apabila tidak memenuhi syarat kelulusan, Anda bisa mengulang pengerjaan kuis kembali.

Selamat Mengerjakan!

# [Functional Programming](https://www.dicoding.com/academies/256/tutorials/15850?from=13952)

## Pengenalan Functional Programming

Seperti yang sudah disampaikan di awal, JavaScript adalah bahasa yang mendukung multiparadigm. Artinya, selain merupakan bahasa pemrograman berorientasi objek, penulisan syntax JavaScript juga menggunakan gaya functional programming.

Sebelumnya kita telah mempelajari tentang paradigma OOP, kali ini kita akan melanjutkan dengan pendekatan paradigma yang lain, yakni Functional Programming.

Kita juga telah membahas mengenai Object Composition. Di mana konsep ini cukup membantu ketika membuat sebuah kelas tanpa harus melakukan proses pewarisan. Ia akan memecah-mecah fungsi yang umum digunakan agar dapat digunakan kembali (reusable) tanpa proses pewarisan.

Nah, Functional Programming adalah pendekatan yang lebih dalam dari konsep tersebut.

Di modul kali ini, kita akan belajar banyak mengenai paradigma functional programming. Hingga di akhir modul ini Anda diharapkan dapat:

* Mengetahui Paradigma Functional Programming.
* Mengetahui konsep Pure Function, Immutability, dan Higher-Order Function.
* Menggunakan Reusable Function yang ada pada JavaScript.

## Paradigma Functional Programming

Paradigma Functional Programming adalah paradigma pemrograman di mana proses komputasi didasarkan pada fungsi matematika murni. Functional Programming (selanjutnya akan kita singkat menjadi FP) ditulis dengan gaya deklaratif yang berfokus pada “what to solve” dibanding “how to solve” yang dianut oleh gaya imperatif.

Sebagai gambaran buat Anda yang belum tahu apa itu deklaratif dan imperatif lebih jauh, silakan simak contoh kode berikut.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13957#run1-editor1)



1

const names = ['Harry', 'Ron', 'Jeff', 'Thomas'];

2

​

3

const newNamesWithExcMark = [];

4

​

5

for(let i = 0; i < names.length; i++) {

6

newNamesWithExcMark.push(`${names[i]}!`);

7

}

8

​

9

console.log(newNamesWithExcMark);

10

​

11

/\* output:

12

[ 'Harry!', 'Ron!', 'Jeff!', 'Thomas!' ]

13

\*/

 Input  Reset

 Jalankan

Contoh kode di atas merupakan salah satu gaya penulisan kode imperatif. Di mana proses pengisian nilai array baru (newNames) berdasarkan array lama (names) dilakukan secara manual. Inilah maksud dari “how to solve”, di mana kita perlu memikirkan bagaimana cara melakukan perulangannya (for); kapan perulangannya harus berhenti (i < names.length); bagaimana cara memasukkan nilai baru ke dalam array (newNamesWithExcMark.push). Huft, sangat melelahkan!

Lantas bagaimana dengan gaya deklaratif? Mari kita lihat kode dengan fungsi yang sama namun dengan gaya deklaratif.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13957#run2-editor1)



1

const names = ['Harry', 'Ron', 'Jeff', 'Thomas'];

2

​

3

const newNamesWithExcMark = names.map((name) => `${name}!`);

4

​

5

console.log(newNamesWithExcMark);

6

​

7

/\* output:

8

\* [ 'Harry!', 'Ron!', 'Jeff!', 'Thomas!' ]

9

\*/

 Input  Reset

 Jalankan

Kesan apa yang pertama Anda rasakan ketika melihat kode di atas? Coba bandingkan dengan kode sebelumnya, tentu ini jauh lebih mudah dibaca dan ringkas. Yap! Inilah yang disebut dengan gaya deklaratif. Kita tidak perlu pusing untuk memikirkan cara manual untuk mencapai sebuah tujuan. Tidak ada proses looping manual; Tidak perlu tahu kapan harus berhenti dari looping; Kita cukup fokus pada “what to solve” alias apa yang ingin kita selesaikan atau capai.

JavaScript sendiri merupakan bahasa pemrograman yang mendukung paradigma FP. Banyak Higher-Order Function (kita akan bahas detail tentang ini nanti) yang bisa kita manfaatkan sebagai utilitas, salah satunya fungsi array map() di atas.

Namun FP bukan hanya sekedar menggunakan High-Order Function bawaan saja. Untuk memahami paradigma FP secara mendalam, kita perlu tahu dulu konsep-konsep apa saja yang ada di dalamnya.

## Konsep-Konsep Functional Programming

Setelah Anda memiliki sedikit gambaran tentang bagaimana paradigma FP dapat membuat kode yang Anda tuliskan lebih mudah dibaca dan ringkas, kini saatnya Anda ketahui, apa saja konsep-konsep dasar yang ada di dalam paradigma FP.

Di sini kita akan membahas 4 konsep besar yang ada di FP. Yakni Pure Function, Immutability, Recursive, dan High-Order Function.

### Pure Function

Salah satu konsep besar dari paradigma FP adalah Pure Function. Apa artinya? Pure Function merupakan konsep dari pembuatan fungsi yang mengharuskan fungsi untuk **tidak bergantung terhadap nilai yang berada di luar fungsi atau parameternya**. Sehingga mau seperti apa keadaanya, fungsi yang dibuat selalu menghasilkan sesuatu yang sama, terkecuali bila fungsi tersebut diberikan nilai parameter yang berbeda.

Untuk lebih jelasnya, simak kode berikut:

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13958?from=13957#run1-editor1)



1

let PI = 3.14;

2

​

3

const hitungLuasLingkaran = (jariJari) => {

4

return PI \* (jariJari \* jariJari);

5

}

6

​

7

console.log(hitungLuasLingkaran(4)); // 50.24

8

​

9

PI = 5; // tidak sengaja nilai PI berubah

10

​

11

console.log(hitungLuasLingkaran(4)); // 80

 Input  Reset

 Jalankan

Menurut Anda, apakah fungsi hitungLuasLingkaran merupakan pure function atau impure function (lawan dari pure function)? Jika Anda menjawab impure function, Anda tepat sekali!

Fungsi tersebut tidak bisa dikatakan pure function karena ia membutuhkan nilai yang berada di luar dari fungsinya, yakni nilai PI. Bila nilai PI berubah, maka penggunaan fungsi menghasilkan nilai yang berbeda walaupun diberikan nilai parameter yang sama.

Lantas, bagaimana cara membuat fungsi tersebut menjadi pure? Mudah!

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13958?from=13957#run2-editor1)



1

const hitungLuasLingkaran = (jariJari) => {

2

return 3.14 \* (jariJari \* jariJari);

3

}

4

​

5

console.log(hitungLuasLingkaran(4)); // 50.24

6

console.log(hitungLuasLingkaran(4)); // 50.24

7

console.log(hitungLuasLingkaran(8)); // 200.96

8

console.log(hitungLuasLingkaran(8)); // 200.96

 Input  Reset

 Jalankan

Dengan memindahkan nilai PI secara konstan di dalam fungsi, maka dapat dipastikan nilai tersebut tidak akan berubah atau tak sengaja diubah. Dengan begitu fungsi selalu menghasilkan nilai yang sama bila nilai parameter yang diberikan sama.

Selain dilarang untuk bergantung terhadap nilai luar, pure function juga **dilarang keras untuk mengubah nilai yang berada di luar baik secara sengaja atau tidak sengaja**. Pure function tidak boleh menimbulkan efek samping (no side effect) ketika digunakan.

Untuk lebih jelasnya, coba lihat contoh kode berikut:

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13958?from=13957#run3-editor1)



1

const createPersonWithAge = (age, person) => {

2

person.age = age;

3

return person;

4

};

5

​

6

const person = {

7

name: 'Bobo'

8

};

9

​

10

const newPerson = createPersonWithAge(18, person);

11

​

12

console.log({

13

person,

14

newPerson

15

});

16

​

17

/\*\*

18

\* Output:

19

\* {

20

person: { name: 'Bobo', age: 18 },

 Input  Reset

 Jalankan

Fungsi createPersonWithAge bertujuan untuk membuat objek person baru dengan tambahan properti age dari objek person yang ada. Namun alih-alih hanya membuat objek baru, ia juga malah mengubah nilai dari objek lama. Nah, hal inilah yang menyebabkan fungsi createPersonWithAge bukanlah pure function.

Lalu bagaimana cara membuat fungsi tersebut menjadi pure? Mudah, kita manfaatkan destructuring object daripada mengubah objek tersebut secara langsung.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13958?from=13957#run4-editor1)



1

const createPersonWithAge = (age, person) => {

2

return { ...person, age };

3

};

4

​

5

const person = {

6

name: 'Bobo'

7

};

8

​

9

const newPerson = createPersonWithAge(18, person);

10

​

11

console.log({

12

person,

13

newPerson

14

});

15

​

16

/\*\*

17

\* Output:

18

\* {

19

\* person: { name: 'Bobo' },

20

\* newPerson: { name: 'Bobo', age: 18 }

21

\* }

22

\*/

23

​

24

​

 Input  Reset

 Jalankan

Lihat! Lagi-lagi lebih mudah dan singkat bukan?

Agar lebih mudah dalam mengetahui apakah fungsi yang Anda buat sudah pure atau belum, pastikan 3 konsep ini ada pada fungsi yang Anda buat.

* Mengembalikan nilai yang sama bila inputannya (nilai parameter) sama.
* Hanya bergantung pada argumen yang diberikan.
* Tidak menimbulkan efek samping.

Bila 3 konsep di atas terpenuhi, maka bisa dipastikan Anda membuat sebuah pure function.

### Immutability

Konsep yang kedua adalah immutability. Immutable berarti sebuah objek tidak boleh diubah setelah objek tersebut dibuat. Kontras dengan mutable yang artinya objek boleh diubah setelah objek tersebut dibuat.

Konsep immutability sangat kental pada paradigma FP. Anda bisa lihat sebelumnya pada contoh penggunaan array map. Ketika menggunakan array.map(), alih-alih ia mengubah nilai dari array itu sendiri, malah ia membuat atau menghasilkan array baru.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13958?from=13957#run5-editor1)



1

const names = ['Harry', 'Ron', 'Jeff', 'Thomas'];

2

​

3

const newNamesWithExcMark = names.map((name) => `${name}!`);

4

​

5

console.log({

6

names,

7

newNamesWithExcMark,

8

});

9

​

10

/\*\*

11

\* {

12

names: [ 'Harry', 'Ron', 'Jeff', 'Thomas' ],

13

newNamesWithExcMark: [ 'Harry!', 'Ron!', 'Jeff!', 'Thomas!' ]

14

}

15

\*/

16

​

17

​

 Input  Reset

 Jalankan

Lantas, bagaimana bila kita benar-benar perlu mengubah nilai dari sebuah objek? Contohnya seperti ini:

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13958?from=13957#run6-editor1)



1

const user = {

2

firstname: 'Harry',

3

lastName: 'Protter', // ups, typo!

4

}

5

​

6

const renameLastNameUser = (newLastName, user) => {

7

user.lastName = newLastName;

8

}

9

​

10

renameLastNameUser('Potter', user);

11

​

12

console.log(user);

13

​

14

/\*\*

15

\* output:

16

\* { firstname: 'Harry', lastName: 'Potter' }

17

\*

18

\*/

 Input  Reset

 Jalankan

Yup! Tujuan Anda memang tercapai namun itu bukanlah konsep dari paradigma FP. Bila Anda ingin menerapkan FP sepenuhnya, hindari cara seperti di atas. Lantas bagaimana solusinya? Sama seperti fungsi array map(), alih-alih mengubah nilai objek secara langsung, terapkan perubahan tersebut pada nilai return dalam objek baru.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13958?from=13957#run7-editor1)



1

const user = {

2

firstname: 'Harry',

3

lastName: 'Protter', // ups, typo!

4

}

5

​

6

const createUserWithNewLastName = (newLastName, user) => {

7

return { ...user, lastName: newLastName }

8

}

9

​

10

const newUser = createUserWithNewLastName('Potter', user);

11

​

12

console.log(newUser);

13

​

14

/\*\*

15

\* output:

16

\* { firstname: 'Harry', lastName: 'Potter' }

17

\*

18

\*/

 Input  Reset

 Jalankan

Hasilnya sama kan? Selain itu, Anda juga bisa menyesuaikan nama fungsinya dari renameLastNameUser menjadi createUserWithNewLastName. Hal itu perlu untuk mengindikasikan bahwa fungsi mengembalikan atau menghasilkan objek user baru.

### Rekursif

Konsep selanjutnya yang ada di Functional Programming adalah rekursif. Apa itu rekursif? Rekursif merupakan teknik pada sebuah function yang memanggil dirinya sendiri.

Kita akan mencoba mengubah fungsi countDown yang biasanya kita buat menggunakan sintaksis iterasi seperti for, foreach, while seperti kode di bawah menjadi bentuk rekursif.

1. const countDown = start => {
2. do {
3. console.log(start);
4. start -=1;
5. }
6. while(start > 0);
7. };
9. countDown(10);

Maka, bentuk rekursinya adalah sebagai berikut:

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13958?from=13957#run8-editor1)



1

const countDown = start => {

2

console.log(start);

3

if(start > 0) countDown(start-1);

4

};

5

​

6

countDown(10);

 Input  Reset

 Jalankan

Dengan teknik rekursif ini, kita sebenarnya bisa menggantikan operasi iterasi dengan rekursi. Namun tidak sebatas itu saja, dengan rekursi kita dapat membuat dan mengolah data structures seperti Tree dan Array.

### Higher-Order Function

JavaScript memiliki kemampuan [First Class Functions](https://developer.mozilla.org/en-US/docs/Glossary/First-class_Function), karena itu fungsi pada JavaScript dapat diperlakukan layaknya sebuah data. Kita bisa menyimpan function dalam variabel, memberikan function sebagai parameter pada fungsi lainnya, hingga mengembalikan function di dalam function.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13958?from=13957#run9-editor1)



1

const hello = () => {

2

console.log('Hello!')

3

};

4

​

5

const say = (someFunction) => {

6

someFunction();

7

}

8

​

9

const sayHello = () => {

10

return () => {

11

console.log('Hello!');

12

}

13

}

14

​

15

hello();

16

say(hello);

17

sayHello()();

18

​

19

/\*\*

20

\* Hello!

 Input  Reset

 Jalankan

Karena dengan kemampuan First Class Functions-nya itu, kita dapat membuat Higher-Order Function secara mudah. Tunggu, tunggu. Apa itu Higher-Order Function?

Higher-Order Function menjadi bagian konsep pada paradigma FP. Higher-Order Function merupakan fungsi yang dapat menerima fungsi lainnya pada argumen; mengembalikan sebuah fungsi; atau bahkan keduanya.

Teknik Higher-Order Function biasanya digunakan untuk:

* Mengabstraksi atau mengisolasi sebuah aksi, event, atau menangani alur asynchronous menggunakan callback, promise, dan lainnya.
* Membuat utilities yang dapat digunakan di berbagai tipe data.
* Membuat teknik [currying](https://javascript.info/currying-partials) atau function composition.

Array map() merupakan salah satu contoh Higher-Order Function yang ada di JavaScript. Karena dalam penggunaanya, **ia menerima satu buah argumen yang merupakan sebuah function**.

Dengan mengetahui adanya Higher-Order Function, Anda bisa membuat fungsi map() versi Anda sendiri!

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13958?from=13957#run10-editor1)



1

const names = ['Harry', 'Ron', 'Jeff', 'Thomas'];

2

​

3

const arrayMap = (arr, action) => {

4

const loopTrough = (arr, action, newArray = [], index = 0) => {

5

const item = arr[index];

6

if(!item) return newArray;

7

return loopTrough(arr, action, [...newArray, action(arr[index])], index + 1);

8

}

9

​

10

return loopTrough(arr, action);

11

}

12

​

13

​

14

const newNames = arrayMap(names, (name) => `${name}!` );

15

​

16

console.log({

17

names,

18

newNames,

19

});

20

​

 Input  Reset

 Jalankan

## Reusable Function

Dengan menerapkan konsep-konsep yang ada di dalam paradigma FP, fungsi yang Anda buat akan bersifat reusable. Karena fungsi yang Anda buat merupakan pure function, ia tidak akan dipengaruhi ataupun mempengaruhi keadaan di/dari luar. Hal ini tentu membuat fungsi dapat digunakan berkali-kali tanpa khawatir mendapatkan hasil di luar ekspektasi Anda.

Pada materi kali ini kita tidak berfokus untuk membuat banyak reusable function di JavaScript. Kami percaya, dengan memahami konsep functional programming secara matang, Anda dapat membuatnya sendiri sesuai kebutuhan.

Alih-alih membuatnya sendiri, kita akan coba membahas dan menggunakan beberapa reusable function yang sudah ada di JavaScript. Khususnya beberapa Higher-Order Function yang dimiliki array, seperti map, filter, dan forEach.

### Array Map

Fungsi array.map() merupakan fungsi bawaan dari array yang sangat berguna dan banyak sekali digunakan. Fungsi ini dapat dipanggil dari sebuah data bertipe array dan menerima satu buah callback function.

1. ['Harry', 'Ron', 'Jeff', 'Thomas'].map(() => { });

Callback function tersebut akan dipanggil sebanyak jumlah panjang array dan akan memiliki akses pada index array sesuai dengan iterasinya.

1. ['Harry', 'Ron', 'Jeff', 'Thomas'].map((**name**) => { });

Seperti yang Anda ketahui di awal, fungsi map akan mengembalikan array baru. Nilai tiap item pada array yang dikembalikan, dihasilkan dari kembalian callback function-nya. Karena callback function dapat mengakses item array, biasanya developer menggunakannya untuk menghasilkan nilai baru.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/15855?from=13958#run1-editor1)



1

const newArray = ['Harry', 'Ron', 'Jeff', 'Thomas'].map((name) => { return `${name}!`});

2

​

3

console.log(newArray);

4

​

5

/\*\*

6

\* [ 'Harry!', 'Ron!', 'Jeff!', 'Thomas!' ]

7

\*

8

\*/

 Input  Reset

 Jalankan

### Array Filter

Sama seperti array.map(), fungsi array.filter() merupakan fungsi bawaan dari data yang bertipe array di JavaScript. Sesuai namanya, fungsi ini sangat berguna untuk melakukan proses penyaringan (filtering) terhadap nilai array yang ada. Bila Anda memiliki kasus di mana Anda ingin menghilangkan beberapa item di array berdasarkan spesifikasi tertentu, fungsi ini sangatlah cocok Anda gunakan.

Cara kerja fungsi ini mirip seperti array.map(). Namun, callback function dari fungsi ini harus mengembalikan boolean. Di mana nilai boolean ini digunakan untuk menentukan apakah item array lolos saring atau tidak.

Sama seperti fungsi map(), fungsi filter() juga akan mengembalikan array yang telah disaring dalam bentuk array baru.

Contoh penggunaan ketika Anda ingin menghilangkan seluruh nilai false pada array:

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/15855?from=13958#run2-editor1)



1

const truthyArray = [1, '', 'Hallo', 0, null, 'Harry', 14].filter((item) => Boolean(item));

2

​

3

console.log(truthyArray);

4

​

5

/\*\*

6

\* output:

7

\* [ 1, 'Hallo', 'Harry', 14 ]

8

\*

9

\*/

 Input  Reset

 Jalankan

Contoh lain, penggunaan filter untuk menyaring array dari objek siswa yang layak mendapatkan beasiswa berdasarkan nilai skor yang didapat.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/15855?from=13958#run3-editor1)



1

const students = [

2

{

3

name: 'Harry',

4

score: 60,

5

},

6

{

7

name: 'James',

8

score: 88,

9

},

10

{

11

name: 'Ron',

12

score: 90,

13

},

14

{

15

name: 'Bethy',

16

score: 75,

17

}

18

];

19

​

20

const eligibleForScholarshipStudents = students.filter((student) => student.score > 85);

 Input  Reset

 Jalankan

### Array Reduce

Sama seperti array.map(), array.reduce() adalah fungsi bawaan dari data yang bertipe array yang digunakan untuk mengeksekusi fungsi reducer pada setiap elemen array dan hanya mengembalikan output satu nilai saja.

1. arr.reduce(callback(accumulator, currentValue, [currentIndex], [array]), [initialValue])
3. // [...] adalah opsional parameter

Callback function dari fungsi ini dapat diolah untuk manipulasi data currentValue dan menyimpannya pada accumulator. Selain itu, fungsi reduce juga memiliki nilai awal yang dapat didefinisikan pada bagian initialValue.

Contoh penggunaannya misalkan ketika kita ingin menjumlahkan total nilai siswa:

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/15855?from=13958#run4-editor1)



1

const students = [

2

{

3

name: 'Harry',

4

score: 60,

5

},

6

{

7

name: 'James',

8

score: 88,

9

},

10

{

11

name: 'Ron',

12

score: 90,

13

},

14

{

15

name: 'Bethy',

16

score: 75,

17

}

18

];

19

​

20

const totalScore = students.reduce((acc, student) => acc + student.score, 0);

 Input  Reset

 Jalankan

### Array some

array.some() merupakan fungsi bawaan dari array yang cukup sering digunakan. Fungsi ini akan menghasilkan nilai boolean.

1. arr.some(callback(element, [index], [array]), [thisArg])
3. // [...] adalah opsional parameter

Nilai yang dihasilkan didasarkan pada pernyataan apakah ada setidaknya satu dari deretan nilai dalam array tersebut lolos berdasarkan kriteria yang kita tuliskan dalam callback function.

Contoh penggunaannya misalkan kita ingin mengetahui apakah dalam deretan angka terdapat angka genap.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/15855?from=13958#run5-editor1)



1

const array = [1, 2, 3, 4, 5];

2

const even = array.some(element => element % 2 === 0);

3

​

4

console.log(even);

5

​

6

/\*\*

7

output true

8

\*\*/

 Input  Reset

 Jalankan

### Array find

Mirip dengan array.some(), array.find() sesuai namanya digunakan untuk mencari apakah di dalam deretan nilai terdapat nilai yang sesuai dengan kriteria yang kita definisikan pada callback function.

Yang membedakan array.find() dengan array.some(), find akan menghasilkan satu nilai dari elemen yang pertama kali ditemukan berdasarkan kriteria tertentu dan akan menghasilkan nilai undefine bila tidak ada kriteria yang terpenuhi pada item array.

1. arr.find(callback(element, [index], [array]), [thisArg]);
3. // [...] adalah opsional parameter

Sebagai contoh, misalkan kita akan mencari siswa dengan nama `James`.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/15855?from=13958#run6-editor1)



1

const students = [

2

{

3

name: 'Harry',

4

score: 60,

5

},

6

{

7

name: 'James',

8

score: 88,

9

},

10

{

11

name: 'Ron',

12

score: 90,

13

},

14

{

15

name: 'Bethy',

16

score: 75,

17

}

18

];

19

​

20

const findJames = students.find(student => student.name === 'James');

 Input  Reset

 Jalankan

### Array sort

array.sort() merupakan fungsi bawaan dari array yang berguna untuk melakukan pengurutan nilai dari sebuah deretan nilai. Secara default, fungsi sort akan mengubah semua nilai dalam deretan menjadi bentuk string dan mengurutkannya secara ascending.

1. arr.sort([compareFunction])
3. // [...] adalah opsional parameter

Contoh sederhananya adalah sebagai berikut:

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/15855?from=13958#run7-editor1)



1

const months = ['March', 'Jan', 'Feb', 'Dec'];

2

months.sort();

3

console.log(months);

4

// output: [ 'Dec', 'Feb', 'Jan', 'March' ]

5

​

6

const array1 = [1, 30, 4, 1000, 101, 121];

7

array1.sort();

8

console.log(array1);

9

// output: [ 1, 1000, 101, 121, 30, 4 ]

 Input  Reset

 Jalankan

Contoh pengurutan di atas didasarkan pada pengurutan bentuk tipe data string. Oleh karena itu, ketika kita ingin mengurutkan sesuai dengan kriteria yang kita inginkan (berdasarkan tanggal ataupun berdasarkan nilai siswa) maka kita perlu membuat compare function tersendiri.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/15855?from=13958#run8-editor1)



1

const array1 = [1, 30, 4, 1000];

2

​

3

const compareNumber = (a, b) => {

4

return a - b;

5

};

6

const sorting = array1.sort(compareNumber);

7

console.log(sorting);

8

​

9

/\*\*

10

output

11

[ 1, 4, 30, 1000 ]

12

\*\*/

 Input  Reset

 Jalankan

Pada compare function, fungsi akan membandingkan 2 nilai yang akan menghasilkan 3 result yaitu negatif (-), 0, dan positif (+).

* Jika, negative maka `a` akan diletakkan sebelum `b`
* Jika, positive maka `b` akan diletakkan sebelum `a`
* Jika, 0 maka tidak ada perubahan posisi.

### Array every

array.every() merupakan fungsi bawaan dari array yang digunakan untuk mengecheck apakah semua nilai dari sebuah array sesuai dengan kriteria yang didefinisikan. Kembalian dari array.every() adalah nilai Boolean.

1. arr.every(callback(element, [index], [array])

Sebagai contoh, kita akan mengecek apakah seorang siswa telah lulus semua uji materi:

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/15855?from=13958#run9-editor1)



1

const scores = [70,85,90];

2

const minimumScore = 65;

3

​

4

const examPassed = scores.every(score => score >= minimumScore);

5

console.log(examPassed);

6

​

7

/\*\*

8

output

9

true

10

\*\*/

 Input  Reset

 Jalankan

### Array forEach

Array forEach merupakan fungsi bawaan dari array yang berfungsi untuk memanggil fungsi callback pada setiap iterasi index array. Berbeda dari fungsi array lain yang sudah kita bahas, fungsi ini tidak mengembalikan nilai apa pun. Jadi fungsi ini secara harfiah hanya berfungsi untuk memanggil fungsi callback-nya saja, tak lebih dari itu.

Melalui fungsi ini, Anda dapat mengubah sintaks perulangan berdasarkan jumlah array secara imperatif menjadi deklaratif.

* [**Cara imperatif**](https://www.dicoding.com/academies/256/tutorials/15855?from=13958#tab1-code1)
* [Cara deklaratif](https://www.dicoding.com/academies/256/tutorials/15855?from=13958#tab1-code2)

1. const names = ['Harry', 'Ron', 'Jeff', 'Thomas'];
3. for(let i = 0; i < names.length; i++) {
4. console.log(`Hello, ${names[i]}!`);
5. }
7. /\*\*
8. \* output:
9. \* Hello, Harry!
10. \* Hello, Ron!
11. \* Hello, Jeff!
12. \* Hello, Thomas!
13. \*
14. \*/

Namun, ketahuilah bahwa ketika menggunakan forEach, kita tidak bisa menggunakan operator break atau continue pada proses perulangan (Anda bisa melakukannya pada perulangan for). Hal ini juga berlaku ketika pada fungsi map dan filter.

1. const names = ['Harry', 'Ron', 'Jeff', 'Thomas'];
3. for(let i = 0; i < names.length; i++) {
4. if(names[i] === 'Jeff') continue; // Bisa!
6. console.log(`Hello, ${names[i]}!`);
7. }
9. names.forEach((name) => {
10. if(name === 'Jeff') continue; // Tidak Bisa!
11. console.log(`Hello, ${name}`);
12. });

## Kuis Coding : Functional Programming

Untuk menguji kemampuan praktikal Anda dalam memahami materi Functional Programming di JavaScript, silakan selesaikan kuis berikut.

Sebelum Anda mengerjakan kuis, ada beberapa hal yang perlu Anda perhatikan.

* Mohon untuk membaca secara seksama perintah, kriteria, atau soal pada komentar kode yang diberi tanda **TODO.**
* Lakukan pengujian pada kode yang Anda tulis contohnya dengan menggunakan console.log() untuk memastikan hasil sesuai dengan yang Anda harapkan. Gunakan tombol **Jalankan**.

Bila sudah yakin dengan pekerjaan yang Anda lakukan, silakan submit kode dengan klik tombol **Submit kode**. Jika kode yang di-submit benar, Anda bisa melanjutkan ke modul berikutnya. Jika tidak, Anda bisa memperbaiki kembali kode sebanyak yang Anda mau.

Batas waktu tunggu untuk mengirimkan kembali perbaikan kode adalah 1 menit. Jadi pastikan kode yang Anda tulis sudah memenuhi perintah, kriteria, atau soal pada kuis kali ini.

Good luck!

|  |
| --- |
| **Mengalami kendala?** Cobalah kunjungi [Forum Diskusi](https://www.dicoding.com/academies/256/discussions?tutorial=15915) untuk mencari solusi atau bertanya langsung kepada mentor kami. |

Selamat, Anda telah menyelesaikan modul ini.  
Kode yang Anda kirimkan telah berfungsi dengan baik.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/15915#run1-editor1)



1

/\*\*

2

\* TODO:

3

\* Buatlah variabel greatAuthors yang merupakan array

4

\* berdasarkan hasil filter() dan map() dari books:

5

\* - Gunakan fungsi filter untuk mengembalikan nilai item books

6

\* yang hanya memiliki nilai sales lebih dari 1000000.

7

\* - Gunakan fungsi map pada books yang sudah ter-filter,

8

\* untuk mengembalikan nilai string dengan format:

9

\* - `${author} adalah penulis buku ${title} yang sangat hebat!`

10

\*

11

\* Catatan: Jangan ubah nilai atau struktur dari books

12

\*/

13

​

14

const books = [

15

{ title: 'The Da Vinci Code', author: 'Dan Brown', sales: 5094805 },

16

{ title: 'The Ghost', author: 'Robert Harris', sales: 807311 },

17

{ title: 'White Teeth', author: 'Zadie Smith', sales: 815586 },

18

{ title: 'Fifty Shades of Grey', author: 'E. L. James', sales: 3758936 },

19

{ title: 'Jamie\'s Italy', author: 'Jamie Oliver', sales: 906968 },

20

{ title: 'I Can Make You Thin', author: 'Paul McKenna', sales: 905086 },

21

{ title: 'Harry Potter and the Deathly Hallows', author: 'J.K Rowling', sales: 4475152 },

22

];

23

​

24

// TODO

25

const greatAuthors= (books.filter((book)=>book.sales>1000000)).map(({ author,title }) => {return`${author} adalah penulis buku ${title} yang sangat hebat!`});

26

console.log(greatAuthors)

27

​

28

/\*\*

29

\* Jangan hapus kode di bawah ini

30

\*/

31

​

32

module.exports = { books, greatAuthors };

## Rangkuman Materi

Anda berada di akhir dari modul Functional Programming. Mari kita uraikan materi yang sudah Anda pelajari untuk mempertajam pemahaman.

* Anda sudah mengenal paradigma Functional Programming.
* Anda sudah mengenal konsep Pure Function, Immutability, Recursive, dan Higher-Order Function pada paradigma Functional Programming.
* Anda sudah mencoba beberapa reusable function yang ada di JavaScript seperti array map, array filter, dan array forEach.

Dengan ringkasan tersebut, diharapkan Anda dapat memahami semua materi yang telah disampaikan. Jika belum, Anda bisa ulas kembali materi yang diberikan pada modul ini. Untuk Anda yang sudah merasa mantap, yuk lanjut ke modul berikutnya!

Aturan

Knowledge Check : Functional Programming (FP) bertujuan untuk menguji pengetahuan Anda tentang materi yang telah dipelajari di modul ini.

Terdapat 2 pertanyaan yang harus dikerjakan dalam kuis ini. Beberapa ketentuannya sebagai berikut:

* Syarat nilai kelulusan : 100%
* Durasi ujian : 5 menit

Apabila tidak memenuhi syarat kelulusan, Anda bisa mengulang pengerjaan kuis kembali.

Selamat Mengerjakan!

# [Setup Environment](https://www.dicoding.com/academies/256/tutorials/13963)

## Pengenalan Setup Environment

Sejauh ini kita telah mempelajari beberapa hal mendasar dalam JavaScript sekaligus menjalankan contoh-contoh kodenya pada platform Dicoding. Selanjutnya bagaimana jika Anda ingin mengembangkan aplikasi lebih lanjut di luar platform Dicoding?

Ada beberapa tools yang perlu Anda siapkan untuk memulai mengembangkan aplikasi dengan komputer atau laptop Anda sendiri.

## Text Editor

Dalam mengembangkan aplikasi, kita akan banyak menuliskan kode. Maka dari itu, tools yang wajib kita miliki adalah sebuah text editor. Beberapa sistem operasi sebenarnya sudah menyediakan text editor bawaan. Contohnya Windows memiliki Notepad, Linux memiliki Text Editors, dan Mac OS memiliki TextEdit. Ketiga aplikasi tersebut bisa kita gunakan untuk belajar membuat aplikasi dengan JavaScript, meskipun masih banyak alternatif text editor lainnya selama masih dapat menyimpan sebuah plain text dengan format **.js**.

Perlu diperhatikan bahwa kode yang kita tulis merupakan sebuah plain text. Pastikan Anda menggunakan text editor yang tepat. Jangan pernah gunakan Microsoft Word untuk menuliskan sebuah kode. Ini penting karena aplikasi tersebut menampilkan teks yang telah diformat atau biasa disebut dengan rich text.

Selain text editor bawaan dari sistem operasi, ada beberapa teks editor lain yang memang dirancang khusus untuk menulis kode pemrograman. Teks editor ini umumnya dilengkapi dengan banyak fitur berguna yang mendukung kita dalam mengembangkan aplikasi.

Tutorial dan contoh di kelas ini selanjutnya akan menggunakan teks editor Visual Studio Code. Visual Studio Code merupakan text editor yang dikembangkan oleh Microsoft. Dalam Visual Studio Code terdapat fitur debugging, Git control, syntax highlighting, code completion, snippets, dan code refactoring. Visual Studio Code tersedia untuk sistem operasi Windows, Mac, maupun Linux. Selain itu, text editor ini bisa kita gunakan secara gratis. Untuk mengunduhnya, silakan kunjungi website berikut: <https://code.visualstudio.com/>.

## Terminal

Pada materi selanjutnya kita akan sering menuliskan command line untuk menjalankan program menggunakan Node.js. Tentunya command line dituliskan dalam sebuah terminal.

Sebenarnya Anda tidak perlu menyiapkan atau mengunduh aplikasi apa pun untuk ini, karena sistem operasi baik Windows, MacOS, dan Linux sudah menyediakan Terminal usungannya masing-masing. Untuk menuliskan command line pada Linux dan MacOS, gunakan aplikasi yang bernama “Terminal.” Pada Windows kita bisa menuliskannya melalui “CMD” atau “PowerShell”.

## JavaScript Runtime

Sesuai yang telah dijelaskan pada modul-modul awal, JavaScript mulanya hanya digunakan pada lingkungan web browser. Saat ini pun browser masih merupakan lingkungan eksekusi yang paling umum untuk kode JavaScript [[4](https://learning.oreilly.com/library/view/javascript-the-definitive/9781491952016/ch01.html#idm46198582059640)]. Lingkungan web browser memungkinkan kode JavaScript untuk menerima inputan dari mouse dan keyboard pengguna. Selain itu, JavaScript juga dapat menampilkan output kepada pengguna melalui HTML dan CSS.

Salah satu cara termudah untuk menjalankan kode JavaScript di lingkungan browser adalah menggunakan browser itu sendiri. Kita dapat menggunakan developer tools yang disediakan oleh browser. Developer tools bisa kita akses melalui shortcut **ctrl + shift + i** atau **klik kanan** -> **Inspect Element**. Setelah itu pilih tab **console**. Developer tools ini dilengkapi dengan interpreter yang akan menjalankan kode yang kita tulis.

Selain browser, terdapat runtime lain yang bisa menjalankan JavaScript, yaitu Node.js. Node.js memungkinkan JavaScript dapat berjalan di berbagai platform, tidak hanya browser. Itulah kenapa JavaScript yang awalnya terkenal sebagai bahasa untuk front-end web, kini mulai merambah juga ke ranah back-end dari website.

Node memberikan akses JavaScript ke seluruh sistem operasi, memungkinkan program JavaScript dapat membaca dan menulis file; mengirim dan menerima data melalui jaringan; serta membuat dan melayani permintaan HTTP.

Selanjutnya kita akan fokus untuk pengembangan program JavaScript pada lingkungan Node. Tentunya kita perlu menginstal Node.js terlebih dulu. Penasaran bagaimana caranya? Kita akan bahas pada materi selanjutnya.

## Node.js

Node.js adalah runtime environment untuk JavaScript yang bersifat open-source dan cross-platform. Dengan Node.js kita dapat menjalankan kode JavaScript di mana pun, tidak hanya terbatas pada lingkungan browser.

Node.js menjalankan V8 JavaScript engine (yang juga merupakan inti dari Google Chrome) di luar browser. Ini memungkinkan Node.js memiliki performa yang tinggi.

Node.js juga menyediakan banyak library/module JavaScript yang membantu menyederhanakan pengembangan aplikasi web. Berikut ini adalah beberapa fitur penting dari Node.js yang menjadikannya pilihan utama dalam pengembangan aplikasi:

* **Asynchronous & Event-driven**  
  Semua API dari Node.js bersifat asynchronous, artinya tidak memblokir proses lain sembari menunggu satu proses selesai. Server Node.js akan melanjutkan ke pemanggilan API berikutnya lalu memanfaatkan mekanisme event notification untuk mendapatkan respon dari panggilan API sebelumnya.
* **Very Fast**  
  Eksekusi kode dengan Node.js sangat cepat karena berjalan pada V8 JavaScript Engine dari Google Chrome.
* **Single Threaded but Highly Scalable**  
  Node.js menggunakan model single thread dengan event looping. Mekanisme ini membantu server untuk merespon secara asynchronous dan menjadikan server lebih scalable dibandingkan server tradisional yang menggunakan banyak thread untuk menangani permintaan.

Node.js dirancang untuk aplikasi dengan proses I/O yang intensif seperti network server atau backend API. Pemrograman dengan multithreading relatif lebih berat dan sulit untuk dilakukan. Jika kita ingin membuat web server yang bisa menangani ratusan request bersamaan, menggunakan ratusan thread akan membutuhkan memori yang besar. Oleh karena itu, karakteristik Node yang asynchronous dan single thread dirancang untuk memungkinkan implementasi server yang dapat menangani banyak request pada waktu yang sama.

Bagaimana jika langsung mencoba Node.js pada perangkat kita? Kita akan mulai dengan langkah instalasi Node.js pada materi berikutnya.

## Instalasi Node.js dengan NVM

Selain dengan browser, agar dapat menjalankan JavaScript pada perangkat lokal, kita perlu menginstal Node.js. Node.js memiliki dua versi rilis, yaitu LTS dan current. LTS merupakan kepanjangan dari Long Term Support. Artinya, versi tersebut mendapat dukungan dalam jangka waktu yang lama, sehingga lebih disarankan menggunakan versi ini. Sementara, versi current berisi fitur-fitur baru yang dirilis untuk Node.js.

Ada dua cara dalam memasang Node.js, yakni:

1. sesuai dengan dokumentasi yang diberikan Node.js pada halaman resminya, dan
2. menggunakan tool bernama Node Version Manager (NVM).

Saat ini, cara yang direkomendasikan dalam memasang Node.js adalah dengan bantuan NVM. Alasanya, NVM banyak menyederhanakan proses dalam pemasangan Node.js. Selain itu, Anda bisa mudah menaikkan atau menurunkan versi Node.js tanpa perlu mengulang proses instalasi.

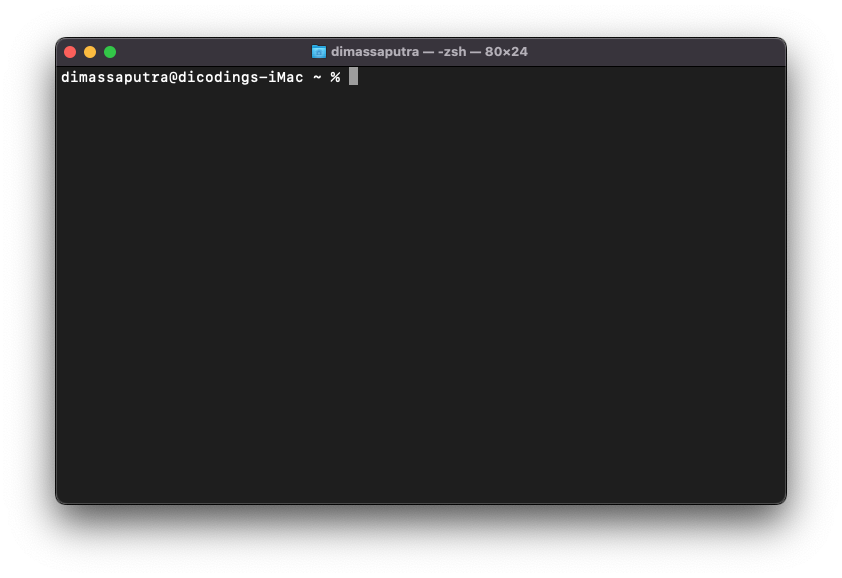
Yuk, ikuti langkah-langkah berikut untuk memasang Node.js dengan NVM sesuai dengan sistem operasi yang Anda gunakan.

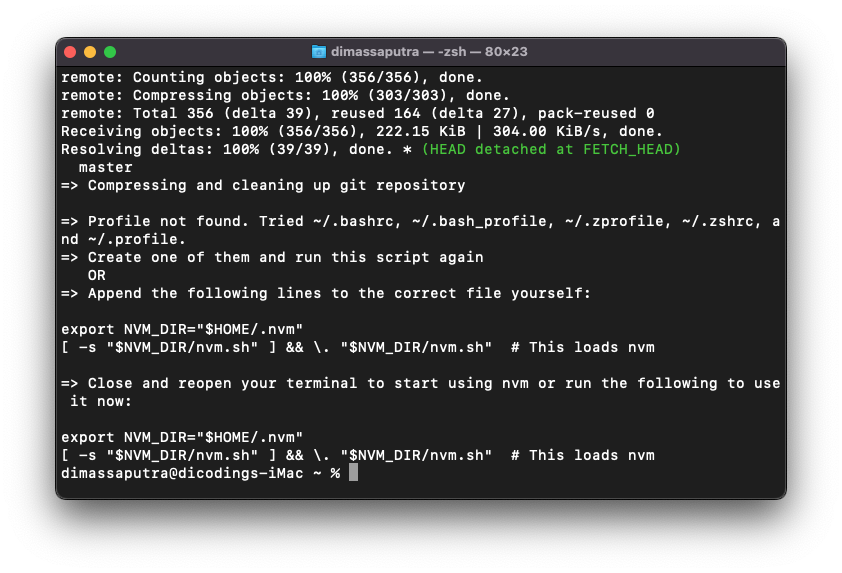
### Linux dan macOS

Pada materi ini kita akan memasang Node.js di Linux dan macOS. Untuk Anda pengguna Windows, gulir materi ini ke bawah hingga menuju bagian Windows.

Seperti yang sudah disebutkan sebelumnya, kita akan memasang NVM, lalu memasang Node.js. Untuk informasi lebih detail mengenai NVM di Linux dan macOS, kunjungi halaman [NVM di GitHub](https://github.com/nvm-sh/nvm).

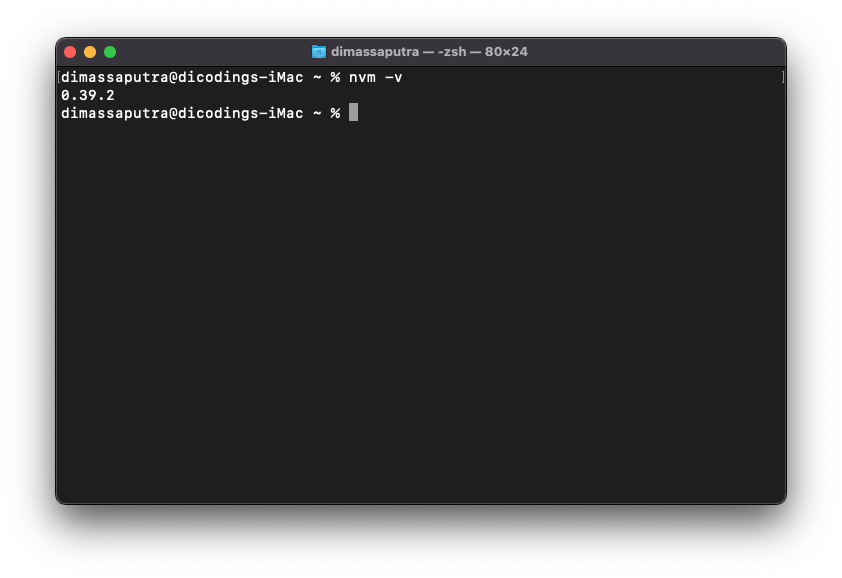
Berikut langkah-langkah dalam memasang NVM di Linux dan macOS.

1. Buka Terminal di macOS atau Linux.  
   [](https://www.dicoding.com/academies/256/tutorials/15830)
2. Kemudian jalankan perintah di bawah ini pada Terminal.
   1. curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.2/install.sh | bash

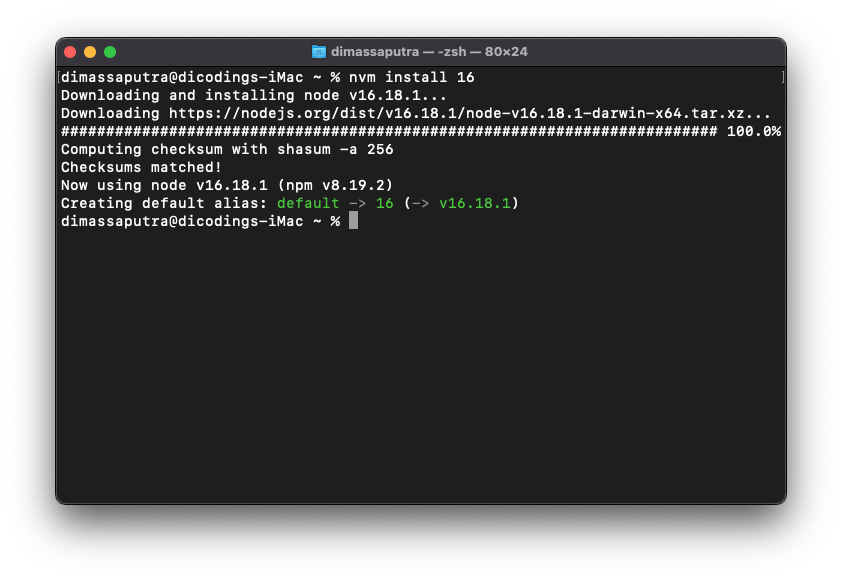
Hasilnya akan tampak seperti gambar di bawah ini.  
[](https://www.dicoding.com/academies/256/tutorials/15830)**Catatan:** Di Linux, jika Anda gagal dalam menjalankan perintah di atas, kemungkinan OS yang Anda gunakan belum terpasang sistem cURL. Jalankan perintah sudo apt install curl, kemudian jalankan kembali kode di atas.  
  
Versi NVM terbaru (per Desember 2022) adalah v.0.39.2. Kami menggunakan versi tersebut pada materi kali ini. Namun, jika Anda ingin memasang versi yang berbeda, Anda bisa mengubah URL dari perintah di atas dengan versi yang Anda inginkan.  
  
Contoh, jika Anda ingin memasang versi v.0.39.1, berarti gunakan perintah berikut.

* 1. curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash

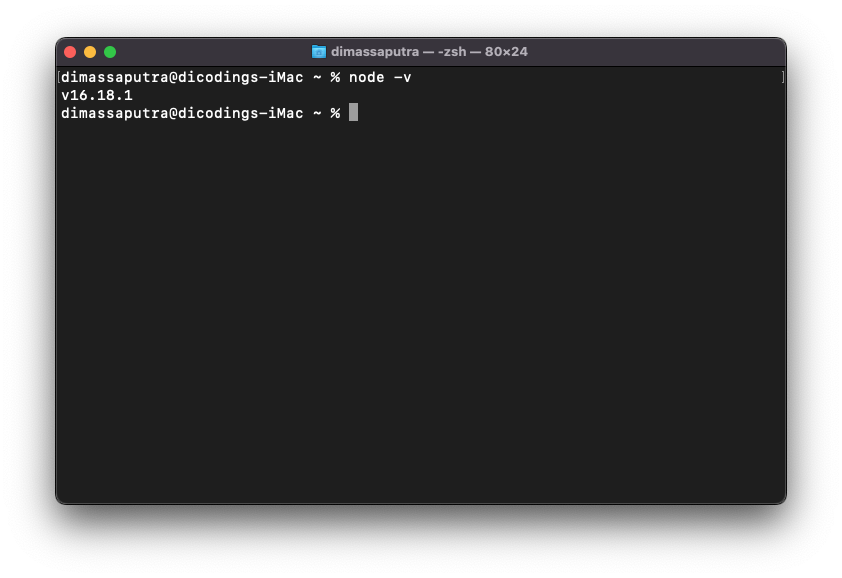
1. Setelah memasang NVM, tutup dan buka kembali Terminal, kemudian jalankan perintah di bawah ini untuk memastikan NVM berhasil terpasang.
   1. nvm -v

Hasilnya akan tampak seperti gambar di bawah ini.  
[](https://www.dicoding.com/academies/256/tutorials/15830)**Catatan:** Jangan khawatir jika perintah NVM masih belum bisa dijalankan. Anda bisa simak penyelesaian masalah umum yang terjadi pada halaman [troubleshooting on macOS](https://github.com/nvm-sh/nvm#troubleshooting-on-macos) atau [troubleshooting on Linux](https://github.com/nvm-sh/nvm#troubleshooting-on-linux).

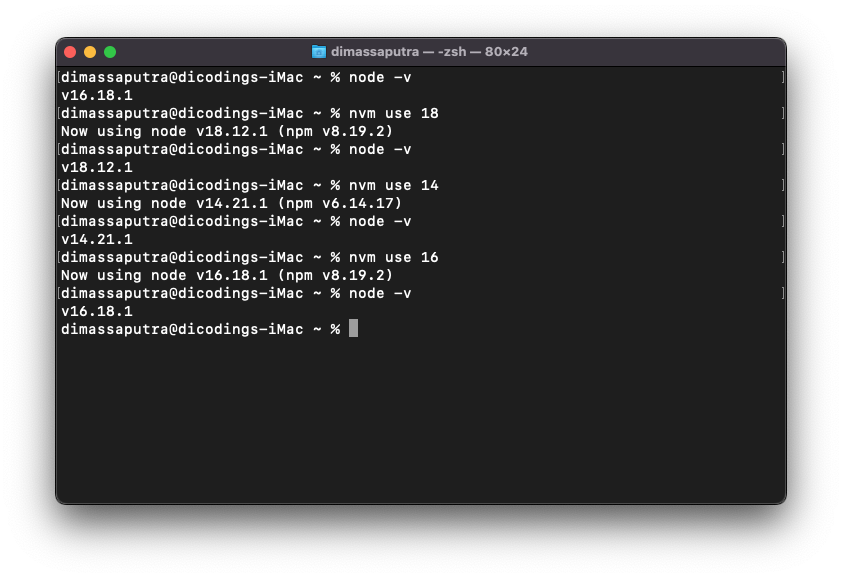
1. Setelah NVM berhasil terpasang, selanjutnya kita pasang Node.js. Cara memasang Node.js adalah dengan perintah berikut ini.
   1. nvm install 16

Perintah tersebut akan memasang Node.js versi 16 terbaru.  
[](https://www.dicoding.com/academies/256/tutorials/15830)Nomor versi pada perintah tersebut bisa Anda ubah sesuai dengan versi yang diinginkan. Contoh, jika ingin memasang Node.js versi 18, gunakan perintah nvm install 18.

1. Untuk memastikan Node.js terpasang dengan baik, jalankan perintah berikut ini.
   1. node -v

Jika berhasil terpasang, hasilnya akan tampak seperti gambar di bawah ini.  
[](https://www.dicoding.com/academies/256/tutorials/15830)

1. Benefit lain yang bisa kita dapatkan dengan menggunakan NVM adalah mudah untuk berpindah-pindah versi Node.js. Untuk pindah versi, Anda cukup tulis perintah berikut ini.
   1. nvm use (versi Node.js)

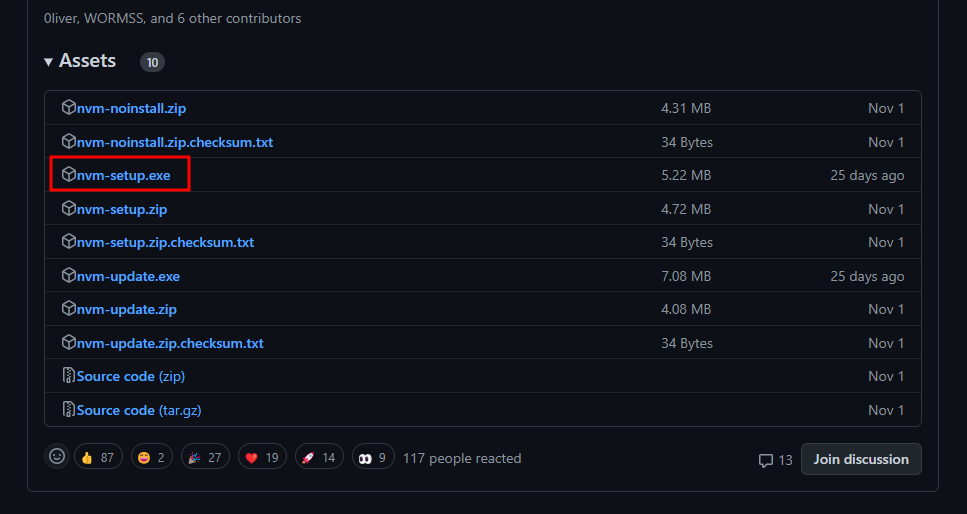
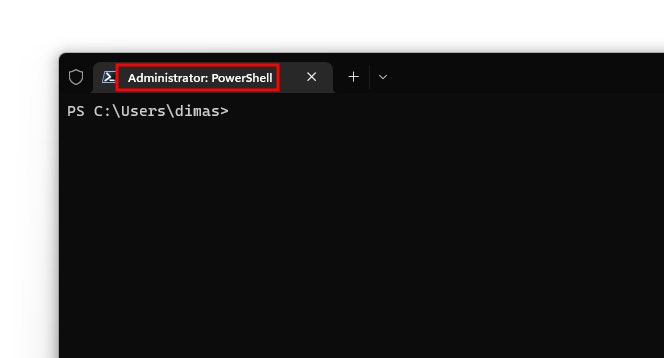
Contohnya seperti gambar di bawah ini.  
[](https://www.dicoding.com/academies/256/tutorials/15830)Agar bisa berpindah versi, pastikan versi tersebut sudah terpasang sebelumnya melalui perintah nvm install (versi Node.js).

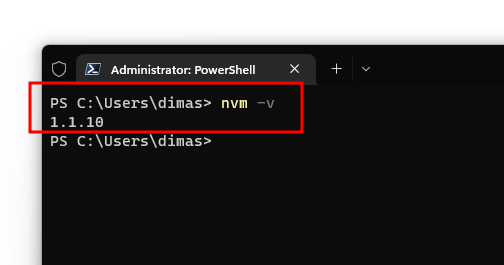
### Windows

Sejatinya [NVM](https://github.com/nvm-sh/nvm) merupakan proyek yang dibuat khusus untuk macOS dan Linux. Namun, jika Anda menggunakan Windows, terdapat alternatif yang tool bernama nvm-windows yang bisa digunakan. Walau namanya mirip, nvm-windows tidak sama dengan nvm yang ada di Linux dan macOS.

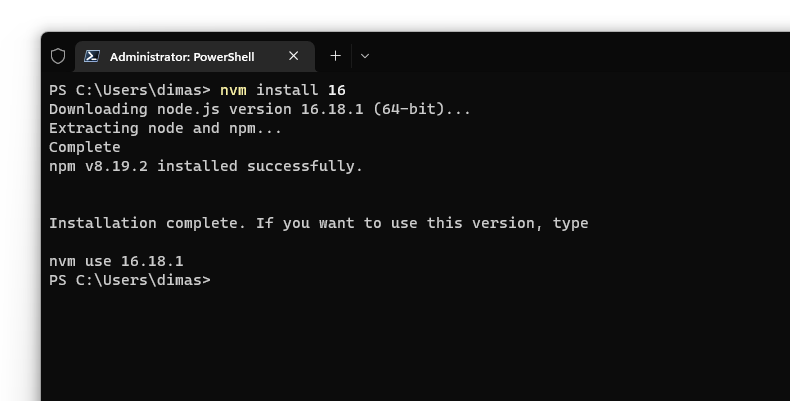
Kita akan memasang NVM di Windows, lalu memasang Node.js. Untuk informasi lebih detail mengenai NVM di Windows, kunjungi halaman [nvm-windows](https://github.com/coreybutler/nvm-windows).

Berikut adalah langkah-langkah dalam memasang nvm-windows.

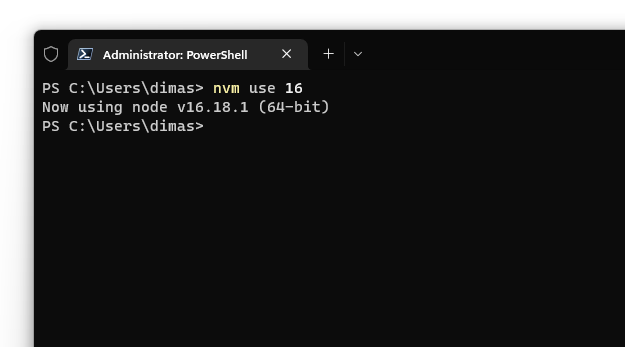
1. Kunjungi halaman unduh nvm-windows pada tautan berikut: <https://github.com/coreybutler/nvm-windows/releases>.
2. Unduh installer NVM (for Windows) dengan klik tautan untuk berkas nvm-setup.exe.  
   [](https://www.dicoding.com/academies/256/tutorials/15830)
3. Setelah berkas installer diunduh, jalankan installer-nya.
4. Lakukan proses instalasi secara default dengan menekan tombol **Next** hingga selesai.
5. Setelah proses instalasi selesai, bukalah Terminal/PowerShell/CMD dalam **mode administrator**.  
   [](https://www.dicoding.com/academies/256/tutorials/15830)
6. Pastikan NVM berhasil terpasang dengan menggunakan perintah berikut ini.
   1. nvm -v

Jika NVM berhasil terpasang, akan menampilkan hasil seperti gambar di bawah ini.  
[](https://www.dicoding.com/academies/256/tutorials/15830)

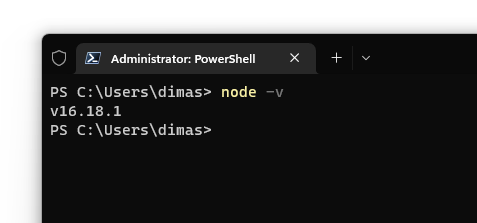
1. Setelah NVM berhasil terpasang, selanjutnya kita pasang Node.js. Cara memasang Node.js adalah dengan perintah berikut ini.
   1. nvm install 16

Perintah tersebut akan memasang Node.js versi 16 terbaru. Prosesnya membutuhkan waktu yang relatif dengan koneksi internet yang Anda miliki.  
[](https://www.dicoding.com/academies/256/tutorials/15830)  
Nomor versi pada perintah tersebut bisa Anda ubah sesuai dengan versi yang diinginkan. Contoh, jika ingin memasang Node.js versi 18, gunakan perintah nvm install 18.

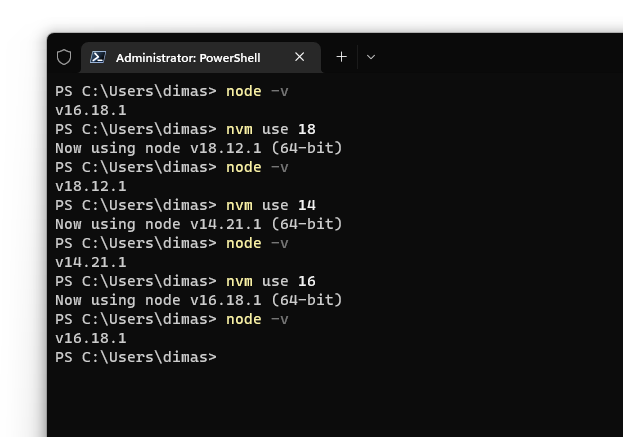
1. Setelah NVM memasang Node.js, Anda bisa menggunakan Node.js dengan menggunakan perintah berikut ini.
   1. nvm use 16

Hasilnya akan tampak seperti gambar di bawah ini.  
[](https://www.dicoding.com/academies/256/tutorials/15830)

1. Untuk memastikan Node.js terpasang dengan baik, jalankan perintah berikut ini.
   1. node -v

Jika berhasil terpasang, hasilnya akan tampak seperti gambar di bawah ini.  
[](https://www.dicoding.com/academies/256/tutorials/15830)

1. Benefit lain yang bisa kita dapatkan dengan menggunakan NVM adalah mudah untuk berpindah-pindah versi Node.js. Untuk pindah versi, Anda cukup tulis perintah berikut ini.
   1. nvm use (versi Node.js)

Contohnya seperti gambar di bawah ini.  
[](https://www.dicoding.com/academies/256/tutorials/15830)  
Agar bisa berpindah versi, pastikan versi tersebut sudah terpasang sebelumnya melalui perintah nvm install (versi Node.js).

## Membuat Project JavaScript

Setelah berhasil menginstal text editor dan juga Node.js di komputer, artinya peralatan “perang” kita sudah siap.

Sebenarnya kita bisa membuat berkas atau project JavaScript secara manual dengan membuat folder dan meletakkan berkas berekstensi .js di dalamnya. Namun, untuk memudahkan pengembangan ke depannya, kita akan membuat proyek menggunakan Node Package Manager (NPM). NPM digunakan untuk mengelola package tambahan untuk mempermudah kita mengembangkan aplikasi. Akan ada modul tersendiri yang membahas lebih rinci terkait NPM. Untuk saat ini, kita baru akan menggunakan NPM untuk membuat proyek Node.js.

Pada direktori komputer Anda, buatlah folder baru sebagai folder utama dari proyek yang akan kita buat. Di sini kita beri nama folder tersebut dengan “CoffeeMachine”. Kemudian buka folder tersebut menggunakan Visual Studio Code. Anda dapat melakukannya dengan memilih menu **File** → **Open Folder …** → Lalu pilih folder project Anda.

Lalu buka terminal/command prompt pada project tersebut dengan memilih menu **Terminal** → **New Terminal**. Ketika terminal muncul jalankan perintah:

1. npm init

Selanjutnya Anda akan diberikan beberapa pertanyaan untuk mengisi nilai package name, version, description, dsb. Semua itu merupakan informasi dasar tentang aplikasi yang kita buat.

Nilai yang berada di dalam tanda kurung merupakan nilai default. Jika nilai default tersebut sudah cocok dengan yang diharapkan, kita dapat menggunakan nilainya dengan langsung menekan tombol enter.

Setelah mengisi seluruh pertanyaan yang diberikan, kita akan diminta untuk melihat dan memverifikasi informasi yang akan disimpan.

Jika nilai yang ditampilkan sudah sesuai, tekan tombol **enter**. Nilai tersebut akan tersimpan dalam berkas **package.json**. Jika, berkas package.json dibuka, hasilnya terlihat seperti gambar di bawah ini.

Berkas tersebut digunakan untuk menyimpan informasi aplikasi yang kita buat. Untuk membuat berkas package.json, sebenarnya kita dapat membuatnya sendiri layaknya membuat berkas baru pada umumnya. Namun, cara tersebut bukan pendekatan yang baik. Dalam membuat berkas package.json, sebaiknya gunakan perintah **npm init** pada Terminal di dalam project yang kita buat.

## Menjalankan Project Node

Pada modul sebelumnya kita telah tahu bagaimana membuat proyek Node.js. Sekarang kita akan mulai menuliskan kode dan menjalankannya melalui terminal.

Pertama, buatlah berkas baru di dalam folder proyek Anda. Caranya, **klik kanan** pada daftar file yang ada di Visual Studio Code lalu pilih **New File**. Beri nama berkas sesuai yang Anda masukkan pada package.json (default-nya adalah **index.js**).

Pada berkas inilah kita bisa mulai menuliskan kode JavaScript. Sebagai permulaan, cetak sejumlah proses pembuatan kopi dari mesin kopi kita. Tambahkan kode berikut ke dalam file index.js:

1. console.log("Menyalakan mesin kopi");
2. console.log("Menggiling biji kopi");
3. console.log("Memanaskan air");
4. console.log("Mencampurkan air dan kopi");
5. console.log("Menuangkan kopi ke dalam gelas");
6. console.log("Menuangkan susu ke dalam gelas");
7. console.log("Kopi Anda sudah siap!");

Untuk menjalankan file JavaScript di atas juga cukup mudah, cukup jalankan perintah berikut pada terminal:

1. node index.js

Node.js akan mengeksekusi setiap baris kode yang kita tulis lalu menampilkannya ke konsol terminal.

### Run Scripts

Pada berkas package.json terdapat beberapa object yang penting untuk kita perhatikan, salah satunya adalah object scripts. Secara default object tersebut akan terbentuk ketika package.json dibuat menggunakan perintah init. Nilai default dari scripts adalah seperti ini:

1. "scripts": {
2. "test": "echo \"Error: no test specified\" && exit 1"
3. },

Object scripts merupakan objek yang mengandung kumpulan script di dalamnya. Script tersebut dapat dijalankan kapan saja pada proyek kita. Untuk menjalankan script, gunakan perintah **npm run <script-name>** yang dapat Anda tulis seperti di bawah ini:

1. npm run test

Dengan menjalankan script test, artinya kita mengeksekusi kode yang berada di dalam nilai test, yaitu:

1. "echo \"Error: no test specified\" && exit 1"

Sehingga, pada terminal akan menghasilkan output seperti berikut:

Pada object scripts biasanya kita menetapkan script yang sering digunakan secara berkala, seperti menjalankan aplikasi (selama proses development), compiling source code ke tahap produksi, atau melakukan testing.

Untuk menetapkan nilai baru pada object scripts, kita tuliskan nama script sebagai properti. Kemudian tuliskan perintah yang akan dieksekusi sebagai nilai dari properti tersebut. Mari kita buat script baru untuk menjalankan kode dari berkas **index.js**.

Pada object scripts, tuliskan nilai baru dengan properti bernama **start**, kemudian tambahkan perintah untuk mengeksekusi berkas sebagai nilainya:

1. "scripts": {
2. "test": "echo \"Error: no test specified\" && exit 1",
3. "start": "node index.js"
4. },

Kini kita bisa menjalankan program dengan perintah npm run start seperti di bawah ini

## Rangkuman Materi

Pada modul ini Anda telah menyiapkan tools yang dibutuhkan untuk mengembangkan aplikasi JavaScript. Dengan ini Anda dapat mulai membuat dan menjalankan aplikasi pada perangkat Anda sendiri tanpa tanpa tergantung platform seperti Dicoding atau lainnya.

Pada modul ini juga Anda telah mempelajari beberapa hal, seperti:

* Perbedaan runtime browser dan node
* Instalasi text editor
* Instalasi Node
* Membuat dan menjalankan project Node.js

Aturan

Knowledge Check : Setup Environment bertujuan untuk menguji pengetahuan Anda tentang materi yang telah dipelajari di modul ini.

Terdapat 2 pertanyaan yang harus dikerjakan dalam kuis ini. Beberapa ketentuannya sebagai berikut:

* Syarat nilai kelulusan : 100%
* Durasi ujian : 5 menit

Apabila tidak memenuhi syarat kelulusan, Anda bisa mengulang pengerjaan kuis kembali.

Selamat Mengerjakan!

# [Module](https://www.dicoding.com/academies/256/tutorials/13982)

## Pengenalan Module

Ketika aplikasi sudah semakin berkembang, kita bisa saja menuliskan seluruh kode hanya dalam satu berkas. Namun, bayangkan Anda harus mencari baris kode yang menimbulkan bug dari ribuan bahkan jutaan baris kode. Sudah terbayang repotnya bukan? Untuk itulah penting untuk membagi project menjadi beberapa berkas berdasarkan tugas dan fungsinya.

Ketika kita membagi kode JavaScript menjadi beberapa berkas, di situlah kita perlu membuat sebuah modul JavaScript. Apa tujuannya? Tak lain adalah untuk menghubungkan berkas JavaScript yang terpisah agar dapat saling digunakan.

Pada materi ini kita akan belajar bagaimana cara implementasi module pada Node.js dan ES6.

## Export & Import

Modul bekerja dengan cara exporting dan importing nilai. Baik itu nilai variabel, fungsi, array, object, atau class agar bisa digunakan pada berkas JavaScript lain. Satu berkas JavaScript terdiri dari satu module yang dapat kita export menjadi lebih dari satu nilai.

Dalam environment Node.js, gunakan perintah module.exports untuk melakukan proses export module. Setiap berkas JavaScript yang berjalan pada Node, memiliki objek module lokal yang memiliki properti exports. Properti tersebut digunakan untuk mendefinisikan nilai apa yang akan diekspor dari berkas tersebut.

Buatlah berkas baru bernama **state.js** pada project Anda. Kode di bawah ini merupakan contoh bagaimana melakukan export nilai menggunakan module.exports.

1. const coffeeStock = {
2. arabica: 100,
3. robusta: 150,
4. liberica: 200
5. }
7. module.exports = coffeeStock;

Kode module.exports = coffeeStock membuat object coffeeStock ditetapkan sebagai nilai dari module.exports. Nilai properti exports ini nantinya bisa di-import dan digunakan pada berkas JavaScript lain.

Jika Anda mencoba melihat nilai module yang ada pada berkas **state.js** dengan menambahkan kode console.log(module) di akhir berkas, maka kita akan melihat object coffeeStock menjadi nilai dari properti exports.

**Catatan:** untuk melihat hasilnya, Anda harus menjalankan dengan perintah node state.js.

1. Module {
2. id: '.',
3. path: '/home/dicoding/Playground/javascript/CoffeeMachine',
4. exports: { arabica: 100, robusta: 150, liberica: 200 },
5. parent: null,
6. filename: '/home/dicoding/Playground/javascript/CoffeeMachine/state.js',
7. loaded: false,
8. children: [],
9. paths: [
10. '/home/dicoding/Playground/javascript/CoffeeMachine/node\_modules',
11. '/home/dicoding/Playground/javascript/node\_modules',
12. '/home/dicoding/Playground/node\_modules',
13. '/home/dicoding/node\_modules',
14. '/home/node\_modules',
15. '/node\_modules'
16. ]
17. }

Lalu bagaimana caranya untuk melakukan import atau menggunakan object yang sudah di-export? Caranya adalah menggunakan method require().

* [**index.js**](https://www.dicoding.com/academies/256/tutorials/15845?from=15840#tab1-code1)
* [state.js](https://www.dicoding.com/academies/256/tutorials/15845?from=15840#tab1-code2)

1. const coffeeStock = require('./state');
3. console.log(coffeeStock);
5. /\* output
6. { arabica: 100, robusta: 150, liberica: 200 }
7. \*/

Dalam inisialisasi variabel coffeeStock (nama variabel bebas kita tentukan), kita gunakan method require() dengan memberikan parameter lokasi berkas **state.js**. Dengan begitu variabel coffeeStock akan memiliki nilai module.exports yang sama pada berkas **state.js**. Setelah mendapatkan nilainya, kita bebas menggunakannya seperti variabel lokal pada umumnya.

1. const coffeeStock = require('./state');
3. const makeCoffee = (type, miligrams) => {
4. if (coffeeStock[type] >= miligrams) {
5. console.log("Kopi berhasil dibuat!");
6. } else {
7. console.log("Biji kopi habis!");
8. }
9. }
11. makeCoffee("robusta", 80);
13. /\* output
14. Kopi berhasil dibuat!
15. \*/

**Tips**: Jika kita menggunakan lokasi yang relatif (dapat berubah/dipindahkan), pastikan awali dengan menuliskan ./. Contohnya, berkas index.js dan state.js berada pada folder yang sama, maka kita cukup menuliskannya dengan ./state.js.

## Export Beberapa Nilai pada Node.js

Pada materi sebelumnya kita telah mengetahui cara export nilai dari suatu berkas JavaScript. Lantas bagaimana jika kita perlu meng-export beberapa nilai sekaligus?

Mari kita contohkan dengan menambahkan variabel isCoffeeMachineReady pada berkas state.js seperti berikut:

1. const coffeeStock = {
2. arabica: 100,
3. robusta: 150,
4. liberica: 200
5. };
7. const isCoffeeMachineReady = true;

Kita tidak bisa melakukan export dari kedua nilai di atas dengan cara seperti di bawah ini:

1. module.exports = coffeeStock;
2. module.exports = isCoffeeMachineReady;

Baris kode kedua berarti kita menginisialisasikan ulang nilai properti module.exports sehingga nilai yang di-export hanya variabel isCoffeeMachineReady.

Solusinya adalah kita tetap mengekspor satu nilai, tetapi kita akan memanfaatkan **object literals** ({ }).

1. module.exports = {coffeeStock, isCoffeeMachineReady};

Jika kita lihat nilai module pada console, nilai dari properti exports merupakan sebuah objek yang menampung nilai dari objek coffeeStock dan variabel isCoffeeMachineReady.

1. Module {
2. id: '.',
3. path: '/home/dicoding/Playground/javascript/CoffeeMachine',
4. exports: {
5. coffeeStock: { arabica: 100, robusta: 150, liberica: 200 },
6. isCoffeeMachineReady: true
7. },
8. parent: null,
9. filename: '/home/dicoding/Playground/javascript/CoffeeMachine/state.js',
10. loaded: false,
11. children: [],
12. paths: [
13. '/home/dicoding/Playground/javascript/CoffeeMachine/node\_modules',
14. '/home/dicoding/Playground/javascript/node\_modules',
15. '/home/dicoding/Playground/node\_modules',
16. '/home/dicoding/node\_modules',
17. '/home/node\_modules',
18. '/node\_modules'
19. ]
20. }

Lalu bagaimana cara import kedua nilai tersebut? Masih ingat dengan materi destructuring object? Pada berkas index.js kita gunakan teknik destructuring object untuk mendapatkan nilai yang di-import seperti ini:

1. const {coffeeStock, isCoffeeMachineReady} = require('./state');
3. console.log(coffeeStock);
4. console.log(isCoffeeMachineReady);
6. /\* output
7. { arabica: 100, robusta: 150, liberica: 200 }
8. true
9. \*/

Namun ingat, ketika menggunakan destructuring object pastikan penamaan variabel lokal sesuai dengan properti objeknya. Jika tidak, maka variabel tersebut akan menghasilkan undefined.

## ES6 Module

Untuk melakukan export dan import module JavaScript, ada dua cara yang bisa kita gunakan. Sebelumnya kita telah mempelajari salah satu cara yaitu format CommonJS yang berjalan di lingkungan Node.js. Sejak ES6, JavaScript memiliki sistem modular secara native. Karena itu, sistem ini dapat dijalankan baik pada environment Node.js maupun browser.

Pada Node.js sebelumnya tidak ada perbedaan antara export satu atau beberapa nilai. Semua nilai yang akan diekspor dijadikan nilai dari properti module.exports. Pada ES6 module, jika kita hanya mengekspor satu nilai pada sebuah berkas JavaScript baik itu primitive value, function, array, object, atau class, kita gunakan keyword export default. Contohnya seperti ini:

1. const coffeeStock = {
2. arabica: 100,
3. robusta: 150,
4. liberica: 200
5. };
7. export default coffeeStock;

Lalu untuk mengimpor nilainya kita dapat menggunakan keyword import … from seperti berikut ini:

1. import coffeeStock from "./state.js";

Berbeda dengan gaya Node.js, kita gunakan keyword import ketika mendeklarasikan variabel yang di-import. Kita juga menggunakan keyword from untuk menentukan lokasi berkas JavaScript-nya.

Ketika menggunakan export default, kita dapat menggunakan penamaan apa saja saat mendeklarasikan variabel untuk menyimpan nilai yang diimpor.

1. import stock from "./state.js";

Hal tersebut aman untuk dilakukan karena dengan menggunakan export default dapat dipastikan hanya satu nilai yang diekspor pada satu berkas JavaScript.

Setelah kita berhasil mendapatkan nilai yang diekspor,kita dapat menggunakan nilainya layaknya variabel lokal.

* [**index.js**](https://www.dicoding.com/academies/256/tutorials/13987#tab1-code1)
* [state.js](https://www.dicoding.com/academies/256/tutorials/13987#tab1-code2)

1. import coffeeStock from './state.js';
3. const displayStock = stock => {
4. for (const type in stock) {
5. console.log(type);
6. }
7. }
9. displayStock(coffeeStock);

Sekarang jalankan kode di atas pada project Anda. Apakah muncul eror seperti berikut?

Jika iya, itu disebabkan karena berkas JavaScript yang kita buat tidak dianggap sebagai module. Saat ini, fitur ES6 module tidak secara default diaktifkan. Pesan eror di atas menyebutkan dua cara bagaimana mengaktifkan ES6 module. Dua cara tersebut adalah menambahkan properti pada **package.json** atau dengan mengubah ekstensi **.js** menjadi **.mjs**. Mari kita gunakan cara pertama yang lebih sederhana.

1. {
2. "name": "coffeemachine",
3. "version": "1.0.0",
4. "description": "",
5. "main": "index.js",
6. "type": "module",
7. "scripts": {
8. "test": "echo \"Error: no test specified\" && exit 1",
9. "start": "node index.js"
10. },
11. "author": "",
12. "license": "ISC"
13. }

Tambahkan properti **type** dengan nilai **module** pada berkas **package.json**. Lalu jalankan kembali program Anda. Seharusnya Node.js sudah berjalan dengan baik dan menampilkan output seperti berikut:

1. arabica
2. robusta
3. liberica

Jika sebelumnya kita hanya melakukan ekspor satu nilai pada berkas JavaScript menggunakan default export, selanjutnya kita akan membahas bagaimana mengekspor banyak nilai dalam satu berkas JavaScript menggunakan ES6.

Named export digunakan untuk mengekspor banyak nilai dalam berkas JavaScript. Cara kerjanya mirip seperti pada Node.js. Nilai yang akan diekspor dituliskan di dalam object literals, seperti ini:

1. const coffeeStock = {
2. arabica: 100,
3. robusta: 150,
4. liberica: 200
5. };
7. const isCoffeeMachineReady = true;
9. export { coffeeStock, isCoffeeMachineReady };

Lalu untuk mendapatkan nilai yang diekspor melalui named export, kita gunakan destructuring object.

1. import { coffeeStock, isCoffeeMachineReady } from './state.js';
3. console.log(coffeeStock);
4. console.log(isCoffeeMachineReady);
6. /\* output
7. { arabica: 100, robusta: 150, liberica: 200 }
8. true
9. \*/

Karena named import menggunakan teknik destructuring object untuk mendapatkan nilai, maka pastikan penamaan variabel sesuai dengan nama variabel yang diekspor. Jika nama tidak sesuai, akan terjadi seperti berikut:

1. import { stock, isCoffeeMachineReady } from './state.js';
3. /\* output
4. SyntaxError: The requested module './state.js' does not provide an export named 'stock'
5. \*/

Namun, jika kita tetap ingin mengubah penamaan variabel dari named import, kita bisa melakukannya dengan menambahkan keyword as setelah penamaan variabelnya.

1. import { coffeeStock as stock, isCoffeeMachineReady } from './state.js';
3. console.log(stock);
4. console.log(isCoffeeMachineReady);
6. /\* output
7. { arabica: 100, robusta: 150, liberica: 200 }
8. true
9. \*/

## Kuis Coding : Module

Untuk menguji kemampuan praktikal Anda dalam memahami materi Module di JavaScript, silakan selesaikan kuis berikut.

Sebelum Anda mengerjakan kuis, ada beberapa hal yang perlu Anda perhatikan.

* Mohon untuk membaca secara seksama perintah, kriteria, atau soal pada komentar kode yang diberi tanda **TODO.**
* Lakukan pengujian pada kode yang Anda tulis contohnya dengan menggunakan console.log() untuk memastikan hasil sesuai dengan yang Anda harapkan. Gunakan tombol **Jalankan**.
* Anda tidak bisa menggunakan ES6 module pada kuis ini, jadi gunakan sistem module CommonJS atau bawaan dari Node.js.

Bila sudah yakin dengan pekerjaan yang Anda lakukan, silakan submit kode dengan klik tombol **Submit kode**. Jika kode yang di-submit benar, Anda bisa melanjutkan ke modul berikutnya. Jika tidak, Anda bisa memperbaiki kembali kode sebanyak yang Anda mau.

Batas waktu tunggu untuk mengirimkan kembali perbaikan kode adalah 1 menit. Jadi pastikan kode yang Anda tulis sudah memenuhi perintah, kriteria, atau soal pada kuis kali ini.

Good luck!

|  |
| --- |
| **Mengalami kendala?** Cobalah kunjungi [Forum Diskusi](https://www.dicoding.com/academies/256/discussions?query=&query_criteria=&sort=&sort_direction=&title=&tutorial=15920&keywords=&creator=&page=1) untuk mencari solusi atau bertanya langsung kepada mentor kami. |

Selamat, Anda telah menyelesaikan modul ini.  
Kode yang Anda kirimkan telah berfungsi dengan baik.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/15920#run1-editor1)
* [Tiger.js](https://www.dicoding.com/academies/256/tutorials/15920#run1-editor2)
* [Wolf.js](https://www.dicoding.com/academies/256/tutorials/15920#run1-editor3)



1

/\*\*

2

\* TODO 1 (Tiger.js):

3

\* Ekspor nilai dari class Tiger

4

\*

5

\* TODO 2 (Wolf.js)

6

\* Ekspor nilai dari class Wolf

7

\*

8

\* TODO 3 (main.js)

9

\* Impor class Tiger dan Wolf

10

\*

11

\* TODO 4 (main.js)

12

\* Ekspor fungsi fight, myTiger, myWolf, dan result

13

\*

14

\*/

15

​

16

​

17

// TODO 3

18

const Tiger = require('./Tiger.js');

19

const Wolf = require('./Wolf.js');

20

​

21

const fight = (tiger, wolf) => {

22

if (tiger.strength > wolf.strength) {

23

return tiger.growl();

24

}

25

if (wolf.strength > tiger.strength) {

26

return wolf.howl();

27

}

28

return 'Harimau dan serigala sama-sama kuat!';

29

};

30

​

31

const myTiger = new Tiger();

32

const myWolf = new Wolf();

33

​

34

const result = fight(myTiger, myWolf);

35

console.log(result);

36

​

37

// TODO 4

38

module.exports = {fight, myTiger, myWolf, result};

 Input  Reset

 Jalankan  Submit Kode

## Rangkuman Materi

Pada modul ini kita telah mempelajari bagaimana membuat beberapa berkas JavaScript bisa saling berkomunikasi menggunakan module. Beberapa poin yang telah kita pelajari, antara lain:

* Bagaimana meng-export dan meng-import nilai antar berkas JavaScript.
* Mempelajari dua jenis format module yang bisa digunakan pada JavaScript.

Aturan

Knowledge Check : Module bertujuan untuk menguji pengetahuan Anda tentang materi yang telah dipelajari di modul ini.

Terdapat 2 pertanyaan yang harus dikerjakan dalam kuis ini. Beberapa ketentuannya sebagai berikut:

* Syarat nilai kelulusan : 100%
* Durasi ujian : 5 menit

Apabila tidak memenuhi syarat kelulusan, Anda bisa mengulang pengerjaan kuis kembali.

Selamat Mengerjakan!

# [Penanganan Eror](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

## [Pengenalan Penanganan Eror](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Ketika mengembangkan sebuah aplikasi, sebagus apa pun kita membuatnya, akan selalu ada kemungkinan terjadi error atau crash. Entah terjadi karena kesalahan kita dalam menyusun logika kode, kesalahan input dari user, atau banyak alasan lainnya.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Ketika aplikasi mengalami eror, aplikasi akan berhenti dan program setelahnya tidak akan dieksekusi.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Sebagai developer, tentunya menjadi tugas kita untuk memastikan aplikasi tetap berjalan bagaimana pun kondisinya. Termasuk mengatasi ketika terjadi eror. Lantas bagaimana?](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Pada modul ini, kita akan mempelajari beberapa hal, seperti:](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

* [Bagaimana menangani error yang mungkin muncul pada aplikasi supaya tidak crash.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
* [Membuat custom error untuk menandai error yang tidak tersedia pada JavaScript.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Sudah siap? Mari kita lanjut ke materi selanjutnya!](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

## [Try and Catch](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Untuk menangani eror pada JavaScript, gunakan try dan catch. Penulisan kode try-catch untuk menangani eror adalah seperti ini:](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

1. [try {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
2. [// kode](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
3. [} catch (error) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
4. [// error handling](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
5. [}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Taruh kode yang berpeluang menimbulkan eror di dalam blok try. Apabila terjadi eror di dalam blok kode try, maka ia akan ditangkap dan ditangani oleh blok kode catch. Sementara, jika tidak terjadi eror pada kode, maka blok catch akan diabaikan.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

1. [try {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
2. [console.log("Awal blok try");](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
3. [console.log("Akhir blok try");](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
4. [} catch (error) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
5. [console.log("Tidak terjadi eror, maka kode ini diabaikan");](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
6. [}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
8. [/\* output](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
9. [Awal blok try](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
10. [Akhir blok try](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
11. [\*/](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Kode di dalam blok try di atas tidak akan menghasilkan eror, sehingga kode di dalam blok catch akan diabaikan dan tidak dijalankan. Berikut ini adalah contoh kode yang menghasilkan eror:](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

1. [try {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
2. [console.log("Awal blok try");   // (1)](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
3. [errorCode;                      // (2)](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
4. [console.log("Akhir blok try");  // (3)](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
5. [} catch (error) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
6. [console.log("Terjadi error!");  // (4)](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
7. [}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
9. [/\* output](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
10. [Awal blok try](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
11. [Terjadi error!](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
12. [\*/](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Baris kode (2) akan menghasilkan eror. Eksekusi kode di dalam blok try akan dihentikan, sehingga baris kode (3) tidak akan tereksekusi. Kemudian kode akan dilanjutkan ke baris (4) atau blok catch.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Selamat! Anda telah berhasil menangani eror dan menghindarkan aplikasi dari crash (Cobalah untuk menghapus sintaks try-catch dan melihat bagaimana aplikasi akan crash). Namun, bagaimana kita bisa tahu apa yang menyebabkan suatu program mengalami eror? Jika ada informasi yang jelas tentunya akan sangat membantu kita atau pengguna nantinya bukan?](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Sekarang perhatikan blok catch. Di sana catch memiliki satu parameter bernama error (nama variabel bisa diubah). Variabel error tersebut merupakan sebuah object yang menyimpan detail informasi dari error yang terjadi.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Object error memiliki beberapa properti utama di dalamnya, yaitu:](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

* **[name](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)**[: Nama error yang terjadi.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
* **[message](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)**[: Pesan tentang detail error.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
* **[stack](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)**[: Informasi urutan kejadian yang menyebabkan error. Umumnya digunakan untuk debugging karena terdapat informasi baris mana yang menyebabkan error.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Sekarang mari kita coba untuk mengubah kode dan menampilkan properti error di atas.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

1. [try {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
2. [console.log("Awal blok try");   // (1)](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
3. [errorCode;                      // (2)](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
4. [console.log("Akhir blok try");  // (3)](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
5. [} catch (error) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
6. [console.log(error.name);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
7. [console.log(error.message);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
8. [console.log(error.stack);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
9. [}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
11. [/\* output](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
12. [Awal blok try](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
13. [ReferenceError](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
14. [errorCode is not defined](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
15. [ReferenceError: errorCode is not defined](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
16. [at file:///home/dicoding/Playground/javascript/CoffeeMachine/error.js:3:5](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
17. [at ModuleJob.run (internal/modules/esm/module\_job.js:152:23)](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
18. [at async Loader.import (internal/modules/esm/loader.js:166:24)](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
19. [at async Object.loadESM (internal/process/esm\_loader.js:68:5)](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
20. [\*/](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Dari informasi di atas, kita bisa tahu bahwa error yang muncul adalah ReferenceError karena errorCode dianggap sebagai sebuah variabel atau nilai yang tidak terdefinisi.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

### [try-catch-finally](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Selain try dan catch, ada satu blok lagi yang ada dalam mekanisme error handling pada JavaScript, yaitu finally. Blok finally akan tetap dijalankan tanpa peduli apa pun hasil yang terjadi pada blok try-catch.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

1. [try {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
2. [console.log("Awal blok try");](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
3. [console.log("Akhir blok try");](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
4. [} catch (error) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
5. [console.log("Baris ini diabaikan");](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
6. [} finally {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
7. [console.log("Akan tetap dieksekusi");](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
8. [}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
10. [/\* output](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
11. [Awal blok try](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
12. [Akhir blok try](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
13. [Akan tetap dieksekusi](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
14. [\*/](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

## [Throwing Errors](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Sekarang kita lihat implementasi try-catch pada kasus yang lebih umum. Perhatikan kode berikut:](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

1. [const json = '{ "name": "Yoda", "age": 20 }';](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
3. [try {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
4. [const user = JSON.parse(json);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
6. [console.log(user.name);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
7. [console.log(user.age);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
8. [} catch (error) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
9. [console.log(error.name);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
10. [console.log(error.message);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
11. [}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Pada kode di atas, fungsi JSON.parse akan melakukan parsing atau konversi dari variabel json (String) menjadi sebuah object. Skenario seperti di atas akan banyak kita temui ketika melakukan request ke API.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Jalankan kode di atas pada text editor Anda. Seharusnya aplikasi berjalan lancar tanpa menimbulkan eror.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Lalu, bagaimana jika json string tidak sesuai dengan format object JavaScript?](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

1. **[const json = '{ bad json }';](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)**
3. [try {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
4. [const user = JSON.parse(json);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
6. [console.log(user.name);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
7. [console.log(user.age);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
8. [} catch (error) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
9. [console.log(error.name);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
10. [console.log(error.message);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
11. [}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

14. [/\* output](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
15. [SyntaxError](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
16. [Unexpected token b in JSON at position 2](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
17. [\*/](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Apabila json tidak sesuai format, maka JSON.parse akan menimbulkan eror. Eror tersebut akan ditangkap oleh blok catch dan kode di dalamnya yang akan dieksekusi.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Lalu, bagaimana jika json yang didapat adalah seperti ini?](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

1. [const json = '{ "age": 20 }';](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
3. [try {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
4. [const user = JSON.parse(json);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
6. [console.log(user.name); // undefined](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
7. [console.log(user.age);  // 20](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
8. [} catch (error) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
9. [console.log(error.name);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
10. [console.log(error.message);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
11. [}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Secara sintaksis, kode di atas tidak menimbulkan eror, sehingga blok catch akan diabaikan. Namun, tidak adanya properti name pada json sebenarnya sama saja dengan eror karena akan berdampak pada jalannya program kita.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Untuk mengatasinya, kita bisa menggunakan throw. Operator ini akan “melemparkan” eror pada program, sehingga eksekusi kode akan masuk pada blok catch. Berikut ini adalah contoh mengimplementasikan throw untuk menimbulkan eror kita sendiri:](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

1. [const json = '{ "age": 20 }';](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
3. [try {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
4. [const user = JSON.parse(json);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
6. [if (!user.name) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
7. [throw new SyntaxError("'name' is required.");](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
8. [}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
10. [console.log(user.name); // undefined](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
11. [console.log(user.age);  // 20](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
12. [} catch (error) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
13. [console.log(`JSON Error: ${error.message}`);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
14. [}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
16. [/\* output](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
17. [JSON Error: 'name' is required.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
18. [\*/](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Ketika properti user.name tidak memiliki nilai, maka program akan menghasilkan SyntaxError. Di dalamnya kita bisa menentukan pesan yang dapat membantu menjelaskan apa eror yang terjadi.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Sekarang kita telah mengetahui ada banyak kemungkinan eror yang bisa muncul dalam sebuah program. Tentunya akan sangat membantu jika kita bisa memberikan pesan yang sesuai kepada pengguna atau kepada diri kita sendiri untuk menemukan kesalahan dengan lebih mudah.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Sekarang anggaplah json sudah sesuai, tetapi ternyata ada eror lain yang terjadi, misalnya karena variabel yang belum terdefinisi.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

1. [const json = '{ "name": "Yoda", "age": 20 }';](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
3. [try {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
4. [const user = JSON.parse(json);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
6. [if (!user.name) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
7. [throw new SyntaxError("'name' is required.");](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
8. [}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
10. [errorCode;](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
12. [console.log(user.name); // Yoda](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
13. [console.log(user.age);  // 20](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
14. [} catch (error) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
15. [console.log(`JSON Error: ${error.message}`);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
16. [}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
18. [/\* output](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
19. [JSON Error: errorCode is not defined](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
20. [\*/](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Eror berhasil ditangani, tetapi konsol tetap menampilkan pesan “JSON Error”, lantas bagaimana kita bisa menampilkan pesan eror sesuai eror yang muncul?](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Jawabannya adalah dengan if statement.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

1. [try {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
2. [// ...](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
3. [} catch (error) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
4. [if (error instanceof SyntaxError) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
5. [console.log(`JSON Error: ${error.message}`);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
6. [} else if (error instanceof ReferenceError) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
7. [console.log(error.message);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
8. [} else {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
9. [console.log(error.stack);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
10. [}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
11. [}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Dengan operator instanceOf, kita bisa mendapatkan tipe dari eror yang terjadi. Dari sana kita bisa membuat percabangan bagaimana cara menangani erornya.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

## [Custom Error](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Setelah menangani eror, pada materi ini kita akan mempelajari bagaimana membuat eror sendiri. Ketika mengembangkan suatu aplikasi, akan ada banyak sekali kemungkinan munculnya eror. Seringkali, kita membutuhkan kelas eror sendiri untuk menunjukkan kesalahan yang spesifik dan tidak tersedia dalam kelas Error bawaan dari JavaScript.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Mari lihat lagi kode kita sebelumnya.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

1. [let json = '{ "age": 30 }';](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
3. [try {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
4. [let user = JSON.parse(json);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
6. [if (!user.name) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
7. [throw new SyntaxError("'name' is required.");](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
8. [}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
10. [console.log(user.name);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
11. [console.log(user.age);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
12. [} catch (error) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
13. [if (error instanceof SyntaxError) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
14. [console.log(`JSON Error: ${error.message}`);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
15. [} else if (error instanceof ReferenceError) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
16. [console.log(error.message);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
17. [} else {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
18. [console.log(error.stack);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
19. [}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
20. [}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Awalnya, JSON.parse akan mengonversi data String menjadi object. Apabila format String tidak sesuai, maka fungsi tersebut akan melemparkan SyntaxError. Meskipun format atau sintaksis dari json string sudah sesuai, tetap ada kemungkinan data di dalamnya tidak lengkap. Saat ini kita masih menggunakan SyntaxError untuk menandai eror akibat data yang tidak lengkap, padahal secara sintaksis tidak ada masalah dari variabel json. Tentunya akan lebih baik jika kita punya Error yang lebih spesifik, bukan?](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Untuk itu kita bisa membuat kelas Error kita sendiri dengan nama dan pesan yang lebih sesuai. Kelas ini merupakan turunan dari kelas Error yang sudah ada. Sebagai contoh, untuk mengecek validasi data dari json, kita bisa membuat kelas Error seperti ini:](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

1. [class ValidationError extends Error {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
2. [constructor(message) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
3. [super(message);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
4. [this.name = "ValidationError";](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
5. [}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
6. [}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Kelas ValidationError memiliki parameter constructor berupa message yang berisi pesan detail terkait erornya. Mari kita lihat penerapannya pada kode sebelumnya.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

1. [class ValidationError extends Error {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
2. [constructor(message) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
3. [super(message);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
4. [this.name = "ValidationError";](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
5. [}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
6. [}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
8. [const json = '{ "age": 30 }';](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
10. [try {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
11. [const user = JSON.parse(json);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
13. **[if (!user.name) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)**
14. **[throw new ValidationError("'name' is required.");](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)**
15. **[}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)**
16. **[if (!user.age) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)**
17. **[throw new ValidationError("'age' is required.");](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)**
18. **[}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)**
20. [console.log(user.name);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
21. [console.log(user.age);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
22. [} catch (error) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
23. [if (error instanceof SyntaxError) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
24. [console.log(`JSON Syntax Error: ${error.message}`);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
25. [}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)**[else if (error instanceof ValidationError) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)**
26. **[console.log(`Invalid data: ${error.message}`);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)**
27. **[}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)**[else if (error instanceof ReferenceError) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
28. [console.log(error.message);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
29. [} else {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
30. [console.log(error.stack);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
31. [}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
32. [}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
34. [/\* output](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
35. [Invalid data: 'name' is required.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
36. [\*/](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Sekarang kode untuk menangani error menjadi lebih baik bukan? Penggunaan instanceOf akan memberikan hasil eror yang lebih detail dan sesuai dengan eror yang terjadi.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

## [Kuis Coding : Penanganan Eror](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Untuk menguji kemampuan praktikal Anda dalam memahami materi Error Handling di JavaScript, silakan selesaikan kuis berikut.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Sebelum Anda mengerjakan kuis, ada beberapa hal yang perlu Anda perhatikan.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

* [Mohon untuk membaca secara seksama perintah, kriteria, atau soal pada komentar kode yang diberi tanda](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)**[TODO.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)**
* [Lakukan pengujian pada kode yang Anda tulis contohnya dengan menggunakan console.log() untuk memastikan hasil sesuai dengan yang Anda harapkan. Gunakan tombol](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)**[Jalankan](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)**[.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Bila sudah yakin dengan pekerjaan yang Anda lakukan, silakan submit kode dengan klik tombol](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)**[Submit kode](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)**[. Jika kode yang di-submit benar, Anda bisa melanjutkan ke modul berikutnya. Jika tidak, Anda bisa memperbaiki kembali kode sebanyak yang Anda mau.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Batas waktu tunggu untuk mengirimkan kembali perbaikan kode adalah 1 menit. Jadi pastikan kode yang Anda tulis sudah memenuhi perintah, kriteria, atau soal pada kuis kali ini.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Good luck!](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

|  |
| --- |
| **[Mengalami kendala?](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)** [Cobalah kunjungi](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)[[Forum Diskusi](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)](https://www.dicoding.com/academies/256/discussions?query=&query_criteria=&sort=&sort_direction=&title=&tutorial=15925&keywords=&creator=&page=1)[untuk mencari solusi atau bertanya langsung kepada mentor kami.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992) |

[Selamat, Anda telah menyelesaikan modul ini.  
Kode yang Anda kirimkan telah berfungsi dengan baik.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

* [**[main.js](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)**](https://www.dicoding.com/academies/256/tutorials/15925?from=14002#run1-editor1)

[](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[71](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[​](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[72](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[const detectTriangle = (a, b, c) => {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[73](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[// TODO 3](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[74](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[75](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[try{](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[76](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[validateNumberInput(a,b,c);](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[77](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[}catch(error){](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[78](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[return error.message;](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[79](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[80](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[​](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[81](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[if (a === b && b === c) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[82](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[return 'Segitiga sama sisi';](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[83](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[84](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[​](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[85](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[if (a === b || a === c || b === c) {](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[86](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[return 'Segitiga sama kaki';](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[87](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[}](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[88](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[​](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[89](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[return 'Segitiga sembarang';](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[90](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[};](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[91](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[​](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[92](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[console.log(detectTriangle(1, false, 1))](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[93](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[​](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[94](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[console.log(detectTriangle('a', 3, 5))](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[95](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[​](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[96](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[console.log(detectTriangle(12, 2, null))](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[97](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[​](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[98](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[/\*\*](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[99](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[\* Jangan hapus kode di bawah ini](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[100](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[\*/](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[101](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[module.exports = { ValidationError, validateNumberInput, detectTriangle };](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[102](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[​](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Input  Reset](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Jalankan  Submit Kode](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

## [Rangkuman Materi](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Kita telah berada di akhir modul Error Handling. Pada modul ini kita mempelajari bagaimana menangani kemungkinan eror pada program agar tidak mengalami crash.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Beberapa poin yang sudah kita bahas di antaranya:](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

* [Penggunaan sintaksis try-catch untuk menangani eror.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
* [Melemparkan eror sendiri yang tidak terdeteksi oleh JavaScript.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
* [Membuat kelas sendiri untuk menandai eror yang tidak tersedia di JavaScript.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Aturan](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Knowledge Check : Penanganan Eror bertujuan untuk menguji pengetahuan Anda tentang materi yang telah dipelajari di modul ini.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Terdapat 2 pertanyaan yang harus dikerjakan dalam kuis ini. Beberapa ketentuannya sebagai berikut:](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

* [Syarat nilai kelulusan : 100%](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)
* [Durasi ujian : 5 menit](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Apabila tidak memenuhi syarat kelulusan, Anda bisa mengulang pengerjaan kuis kembali.](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

[Selamat Mengerjakan!](https://www.dicoding.com/academies/256/tutorials/15800?from=13992)

# [Asynchronous Process](https://www.dicoding.com/academies/256/tutorials/15805?from=14007)

## [Pengantar Asynchronous Process](https://www.dicoding.com/academies/256/tutorials/15805?from=14007)

[Dalam pemrograman, tidak seluruh proses dapat berjalan dengan sangat cepat. Ada beberapa proses yang membutuhkan waktu tunggu, seperti baca tulis berkas, mendapatkan data dari internet, dan proses lainnya. Agar proses yang berjalan lambat tidak menghalangi proses lainnya, kita harus mengetahui cara menjalankan proses yang lambat secara asynchronous.](https://www.dicoding.com/academies/256/tutorials/15805?from=14007)

[Di modul ini, kita akan membahas detail mengenai proses asynchronous di JavaScript. Berikut adalah beberapa poin yang akan dibahas pada modul asynchronous process.](https://www.dicoding.com/academies/256/tutorials/15805?from=14007)

* [Pengenalan Proses Asynchronous di JavaScript.](https://www.dicoding.com/academies/256/tutorials/15805?from=14007)
* [Penggunaan Fungsi setTimeout().](https://www.dicoding.com/academies/256/tutorials/15805?from=14007)
* [Menangani Proses Asynchronous dengan Pola Callback.](https://www.dicoding.com/academies/256/tutorials/15805?from=14007)
* [Menangani Proses Asynchronous dengan Promise.](https://www.dicoding.com/academies/256/tutorials/15805?from=14007)
* [Memahami Penggunaan Static Method dari objek Promise.](https://www.dicoding.com/academies/256/tutorials/15805?from=14007)
* [Menangani Proses Asynchronous dengan Sintaks async dan await.](https://www.dicoding.com/academies/256/tutorials/15805?from=14007)

## [Pengenalan Asynchronous](https://www.dicoding.com/academies/256/tutorials/15805?from=14007)

[Kode pada sebuah program biasanya dijalankan secara berurutan dan saling menunggu hingga prosesnya selesai. Contoh, pada baris pertama kita memanggil fungsi X dan baris kedua kita memanggil fungsi Y, maka fungsi Y hanya akan dijalankan setelah seluruh proses yang berhubungan dengan fungsi X selesai, begitu juga dengan baris kode selanjutnya. Proses yang dijalankan berdasarkan sebuah urutan dan saling menunggu disebut dengan proses synchronous.](https://www.dicoding.com/academies/256/tutorials/15805?from=14007)

[Dalam asynchronous process, kita bisa menjalankan proses yang berada di baris kedua tanpa harus menunggu kode pada baris pertama selesai dieksekusi. Dalam arti lain, kita bisa melakukan lebih dari satu proses sekaligus dalam waktu yang bersamaan. Mengapa hal ini penting? Simak contoh berikut.](https://www.dicoding.com/academies/256/tutorials/15805?from=14007)

[Kode di bawah ini merupakan fungsi untuk mendapatkan menghasilkan bilangan prima sebanyak input dari pengguna. Jika Anda menginput dengan nilai yang besar, contohnya 1 juta, Anda bisa merasakan sedikit delay hingga akhirnya pesan “proses membuat 1000000 bilangan prima selesai”.](https://www.dicoding.com/academies/256/tutorials/15805?from=14007)

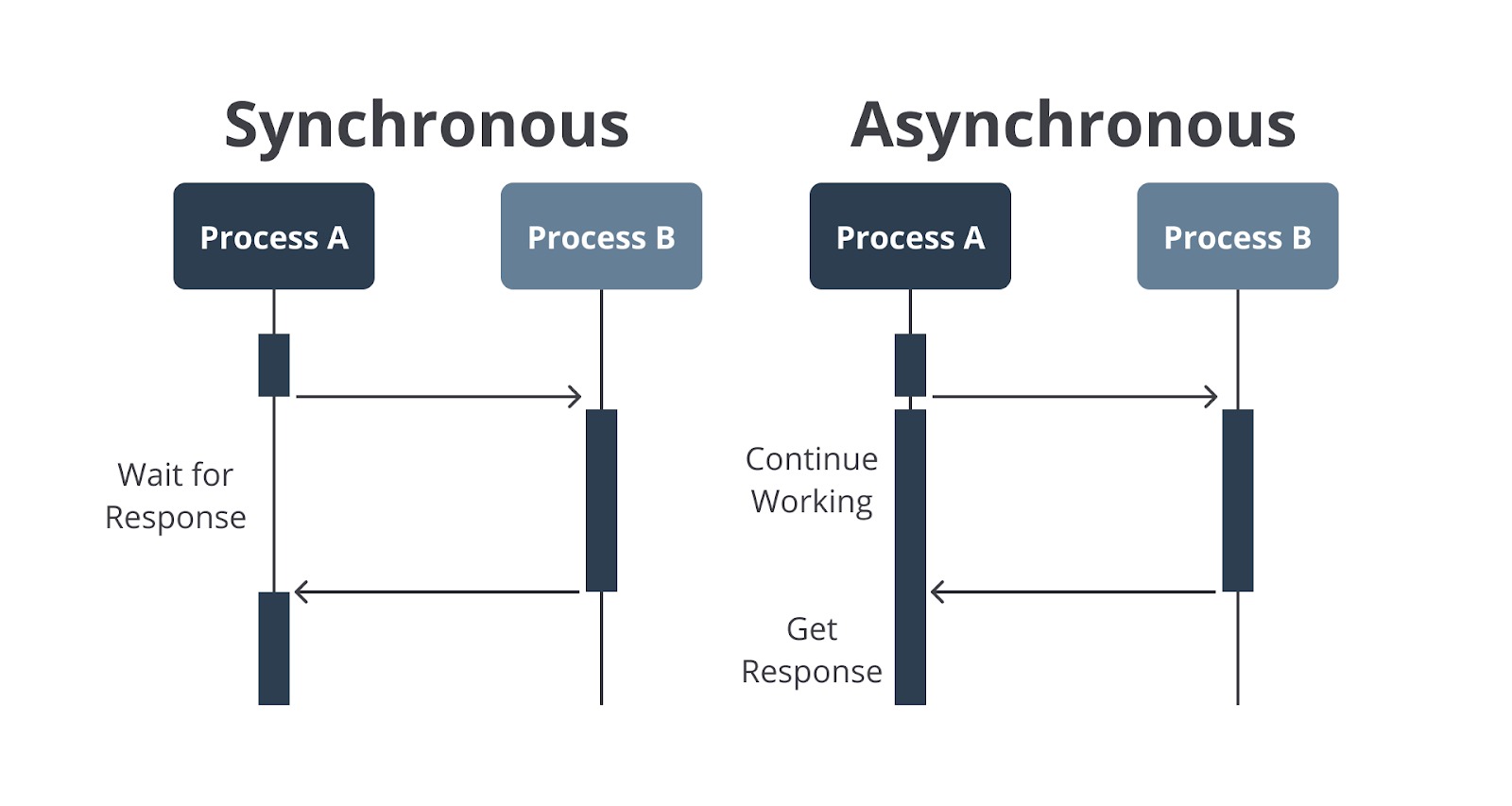
[Delay terjadi karena proses menjalankan fungsi generatePrimes() berjalan lambat dan bersifat synchronous, hingga pengguna merasakan hal itu. Pengalaman ini akan membuat pengguna risi, terlebih ketika pengguna harus melakukan hal lain sembari menunggu proses itu selesai.](https://www.dicoding.com/academies/256/tutorials/15805?from=14007)

[Contoh, cobalah Anda tulis sesuatu pada text area selagi menunggu hasilnya muncul. Apa yang Anda rasakan?](https://www.dicoding.com/academies/256/tutorials/15805?from=14007)

[Perasaan Anda ketika menulis teks pasti tidak nyaman. Proses mendapatkan bilangan prima yang menghalangi Anda untuk menulis teks pada text area tersebut. Ketika proses tersebut selesai, teks di dalam text area akan muncul secara tiba-tiba.](https://www.dicoding.com/academies/256/tutorials/15805?from=14007)

[Itulah jadinya jika proses yang lama dijalankan secara synchronous. Proses selanjutnya perlu menunggu proses sebelumnya selesai. Solusi dari masalah ini adalah dengan menjalankan proses yang lama secara asynchronous.](https://www.dicoding.com/academies/256/tutorials/15805?from=14007)

[Ilustrasi di bawah ini menggambarkan perbedaan synchronous dan asynchronous dalam menjalankan proses.](https://www.dicoding.com/academies/256/tutorials/15805?from=14007)

[[](https://www.dicoding.com/academies/256/tutorials/14012)](https://www.dicoding.com/academies/256/tutorials/15805?from=14007)

[Program asynchronous memungkinkan suatu proses berjalan sembari menunggu proses lainnya selesai. Umumnya, kita memanfaatkan asynchronous pada proses yang besar dan membutuhkan waktu lama, seperti mengambil data dari internet, menyimpan data ke database, dan membaca data dari sebuah berkas.](https://www.dicoding.com/academies/256/tutorials/15805?from=14007)

## setTimeout Function

JavaScript merupakan bahasa pemrograman yang menerapkan pola event-driven, baik pada environment Node.js maupun browser. Seluruh proses yang berjalan dipicu berdasarkan sebuah event atau kejadian, contohnya [DOMContentLoaded](https://developer.mozilla.org/en-US/docs/Web/API/Window/DOMContentLoaded_event) pada browser atau process exit pada Node.js. Dengan pola event-driven itu, artinya kode yang ditulis dengan JavaScript tidak harus dijalankan secara berurutan, tetapi kode dapat dijalankan berdasarkan event atau bahkan dijadwalkan.

Ada satu fungsi di JavaScript yang digunakan untuk menjadwalkan pemanggilan sebuah kode sekaligus membuatnya berjalan secara asynchronous, yakni setTimeout(). Fungsi tersebut menerima dua argumen dengan penjelasan berikut.

1. Argumen pertama merupakan sebuah fungsi yang akan dipanggil secara terjadwal dan asynchronous.
2. Argumen kedua merupakan delay waktu dalam satuan milisecond yang menentukan delay dari pemanggilan fungsi pada argumen pertama.

Inilah contoh dari penggunaan fungsi setTimeout().

1. console.log('Selamat datang!');
3. setTimeout(() => {
4. console.log('Terima kasih sudah mampir, silakan datang kembali!');
5. }, 3000)
7. console.log('Ada yang bisa dibantu?');

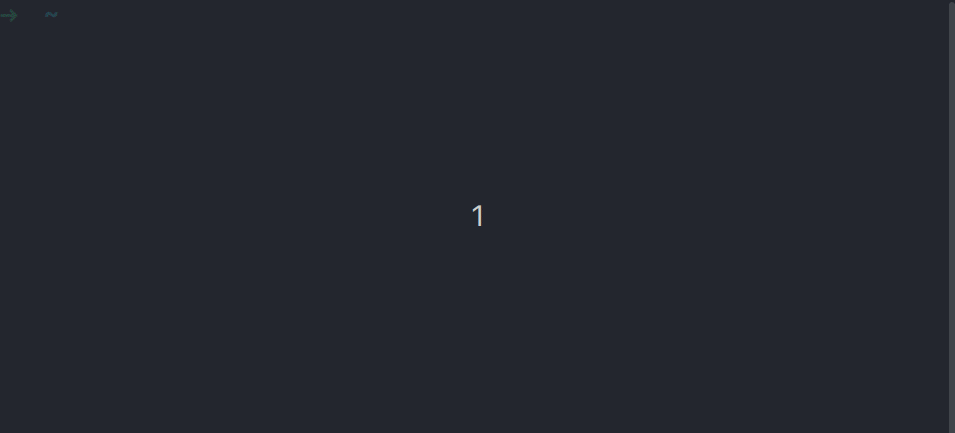
Jika kita terbiasa dengan pola synchronous, wajar bila kita menganggap output pada console memiliki urutan sebagai berikut.

1. Mencetak -> Selamat datang!
2. Menunggu selama tiga detik.
3. Mencetak -> Terima kasih sudah mampir, silakan datang kembali!
4. Mencetak -> Ada yang bisa dibantu?

Namun, nyatanya setTimeout() tidak akan menghentikan JavaScript untuk mengeksekusi kode pada baris berikutnya. Jadi,  urutannya menjadi seperti ini.

1. Mencetak -> Selamat datang!
2. Mencetak -> Ada yang bisa dibantu?
3. Menunggu selama tiga detik.
4. Mencetak -> Terima kasih sudah mampir, silakan datang kembali!

Anda bisa mencobanya dengan menulis kode tersebut di dalam berkas JavaScript kemudian menjalankannya dengan Node.js.

[](https://www.dicoding.com/academies/256/tutorials/15810?from=14012)

Fungsi setTimeout() merupakan cara paling dasar untuk membuat baris kode berjalan secara asynchronous. Kita akan gunakan setTimeout() dalam pembelajaran ini untuk menyimulasikan cara mendapatkan nilai dari sebuah proses yang membutuhkan waktu.

**Catatan:** Ketahuilah bahwa ada beberapa operasi yang berjalan secara asynchronous di berbagai environment JavaScript, contohnya fungsi [fetch()](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API) di browser dan proses input/output pada Node.js. Namun, kita tidak akan menggunakan fungsi-fungsi tersebut pada pembelajaran ini. Seluruh proses asynchronous akan kami simulasikan dengan menggunakan fungsi setTimeout().

## Asynchronous Handling dengan Callback

Di materi sebelumnya kita sudah mengetahui bahwa ada kalanya operasi di JavaScript akan berjalan secara asynchronous. Asynchronous membuat JavaScript dapat mengeksekusi proses selanjutnya tanpa menunggu proses sebelumnya selesai. Perilaku ini membuat penanganan proses asynchronous berbeda dengan synchronous. Sebagai developer, kita harus tahu cara menangani proses asynchronous agar kelak kita bisa mendapatkan nilai yang dihasilkan dari proses tersebut.

Saat ini ada dua cara atau pola yang digunakan JavaScript dalam menangani operasi yang berjalan secara asynchronous. Pola pertama yang akan kita bahas adalah callback.

### Pola Callback

Pola callback merupakan salah satu cara menangani proses asynchronous. Tidak hanya di JavaScript, pola ini juga digunakan pada bahasa pemrograman lainnya. Callback merupakan sebuah fungsi yang disisipkan pada argumen fungsi asynchronous dan akan dipanggil ketika proses dinyatakan selesai. Fungsi callback akan membawa nilai-nilai yang dihasilkan dari proses asynchronous sehingga kita bisa memanfaatkan nilai tersebut.

Jika masih belum terbayang, simak kode di bawah ini.

1. function getUsers(callback) {
2. // simulate network delay
3. setTimeout(() => {
4. const users = ['John', 'Jack', 'Abigail'];
5. callback(users);
6. }, 3000);
7. }

Anggaplah getUsers() merupakan fungsi yang digunakan untuk mendapatkan data nama pengguna yang diambil dari internet. Kami gunakan setTimeout() untuk menyimulasikan bahwa fungsi berjalan secara asynchronous. Anda juga bisa melihat bahwa fungsi tersebut memanggil argumen callback ketika prosesnya selesai dan membawa data users.

Berikut adalah cara mengonsumsi fungsi tersebut untuk mendapatkan nilai users.

1. getUsers(**(users) => {**
2. **console.log(users);**
3. **}**);

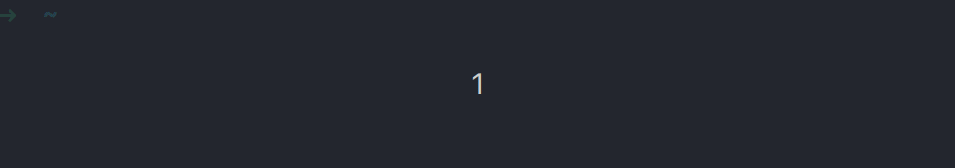
Ketika menggunakan fungsi getUsers() kita perlu memanggil dengan memberikan argumen fungsi. Fungsi ini akan dipanggil ketika proses asynchronous selesai sekaligus membawa data users yang dihasilkan. Di dalam cakupan fungsi inilah kita bisa mengonsumsi nilai users yang dihasilkan dari proses asynchronous. Inilah yang dimaksud dengan pola callback.

**Catatan:** Callback merupakan salah satu implementasi dari konsep High-Order Function yang sudah kita bahas pada modul Functional Programming.

Agar kode lebih mudah dibaca, Anda bisa mendeklarasikan fungsi callback di luar dari argumen seperti ini.

1. function getUsers(callback) {
2. // simulate network delay
3. setTimeout(() => {
4. const users = ['John', 'Jack', 'Abigail'];
6. callback(users);
7. }, 3000);
8. }
10. function usersCallback(users) {
11. console.log(users);
12. }
14. getUsers(usersCallback);

Berikut hasilnya jika fungsi tersebut kami jalankan dengan Node.js.

[](https://www.dicoding.com/academies/256/tutorials/15815?from=15810)

Proses yang dijalankan secara asynchronous, biasanya berpotensi menghasilkan eror. Contohnya, sebuah fungsi yang mengambil data dari internet akan menghasilkan eror ketika jaringan kita bermasalah atau fungsi yang membaca berkas akan menghasilkan error ketika berkas tersebut tidak ditemukan.

Dengan alasan tersebut, pada fungsi callback selain membawa argumen data, ia juga membawa argumen error. Argumen error hanya akan memiliki nilai ketika proses asynchronous gagal atau error.

Untuk lebih jelasnya, mari kita ubah fungsi getUsers() menjadi seperti ini.

1. function getUsers(**isOffline**, callback) {
2. // simulate network delay
3. setTimeout(() => {
4. const users = ['John', 'Jack', 'Abigail'];
6. **if (isOffline) {**
7. **callback(new Error('cannot retrieve users due offline'), null);**
8. **return;**
9. **}**
11. callback(null, users);
12. }, 3000);
13. }

Kami menambahkan argumen isOffline pada fungsi getUsers() untuk menentukan proses asynchronous berhasil atau gagal. Sebab prosesnya berpotensi gagal, nilai yang dibawa oleh fungsi callback harus disesuaikan.

Fungsi callback yang digunakan di JavaScript secara standar (convention) memiliki struktur seperti kode di bawah ini.

1. function callback(error, data) {
2. // logika ketika proses asynchronous selesai
3. }

Argumen pertama dari fungsi callback merupakan error. Argumen ini hanya akan bernilai jika proses asynchronous gagal, sebaliknya biasanya bernilai null. Argumen kedua dari fungsi callback merupakan data yang dibawa ketika prosesnya berhasil. Jika proses asynchronous gagal, dia akan bernilai null.

Selain itu, Anda juga perlu mengingat bahwa convention yang ditetapkan oleh Node.js argumen callback selalu diletakan terakhir. Itulah alasannya kami mendefinisikan argumen isOffline kemudian callback pada fungsi getUsers().

**Catatan.**  
Harap untuk diingat convention tersebut karena banyak proses asynchronous yang terdapat pada Node.js API menggunakan struktur tersebut dalam penggunaan callback, seperti fungsi [fs.readFile()](https://nodejs.org/api/fs.html#fsreadfilepath-options-callback) pada Node.js.

Dengan diubahnya fungsi getUsers(), kita juga perlu menyesuaikan struktur callback-nya. Untuk menangani proses asynchronous pada fungsi getUsers() baik ketika prosesnya berhasil ataupun gagal, maka fungsi usersCallback() perlu diubah menjadi seperti ini.

1. function getUsers(isOffline, callback) {
2. // simulate network delay
3. setTimeout(() => {
4. const users = ['John', 'Jack', 'Abigail'];
6. if (isOffline) {
7. callback(new Error('cannot retrieve users due offline'), null);
8. return;
9. }
11. callback(null, users);
12. }, 3000);
13. }
15. function usersCallback(error, users) {
16. if (error) {
17. console.log('process failed:', error.message);
18. return;
19. }
21. console.log('process success:', users);
22. }
24. getUsers(false, usersCallback); // process success: ['John', 'Jack', 'Abigail']
25. getUsers(true, usersCallback); // process failed: cannot retrieve users due offline

Jika kode di atas kami jalankan dengan Node.js, hasilnya akan tampak seperti ini.

[](https://www.dicoding.com/academies/256/tutorials/15815?from=15810)

Apakah Anda sudah paham dengan pola callback ini? Untuk mengasah pemahaman, cobalah selesaikan tantangan yang kami sediakan di bawah ini.

**Tantangan!**  
Kami menyediakan fungsi asynchronous untuk mendapatkan nilai daftar provinsi di Indonesia dengan nama getProvinces(). Fungsi tersebut akan mengembalikan daftar provinsi pada pemanggilan pertama dan menghasilkan error pada pemanggilan kedua. Fungsi getProvinces() hanya menerima satu argumen yakni fungsi callback yang perlu Anda buat.

Cobalah panggil fungsi tersebut dan tangani proses asynchronous-nya dengan pola callback. Pastikan callback yang Anda buat dapat menangani proses yang gagal dan berhasil.

Jika Anda berhasil menyelesaikan tantangan ini, hasilnya akan tampak seperti ini.

[](https://www.dicoding.com/academies/256/tutorials/15815?from=15810)

**Perhatian**  
Fitur kode interaktif di bawah ini tidak berjalan secara otomatis sehingga untuk mengetahui hasilnya Anda harus menekan tombol restart script yang berada di pojok kanan bawah.

### Callback Hell

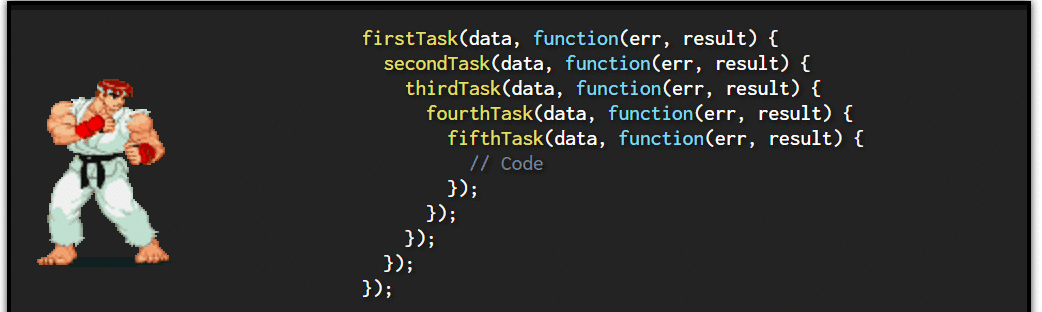
Seiring kompleksnya pengembangan aplikasi, kita akan semakin banyak menggunakan fungsi yang berjalan secara asynchronous. Tidak jarang kita perlu mengagregasikan banyak data dari berbagai proses asynchronous. Masalah akan timbul ketika sebuah proses asynchronous perlu dijalankan setelah proses asynchronous lainnya selesai.

Sebagai developer pemula, kita sangat terbiasa dengan operasi yang saling menunggu pada proses synchronous. Jika Anda coba baca kode di bawah ini, tentu akan mudah memahami alurnya.

Kode di atas menunjukkan sebuah fungsi getUserWeather() yang bertujuan untuk mendapatkan cuaca berdasarkan userId. Anda juga bisa melihat bahwa data cuaca didapatkan dengan cara mengagregasi dua fungsi, yakni getUser() dan getWeather(). Jika kedua fungsi tersebut berjalan secara sinkron, tentu proses agregasi akan mudah karena proses berjalan secara terurut dan saling menunggu.

Sekarang bayangkan jika fungsi getUser() dan getWeather() berjalan secara asinkron dan memanfaatkan pola callback. Untuk menghasilkan output yang sama, kita perlu ubah kode menjadi seperti ini.

Menurut Anda, apakah kode di atas masih nyaman untuk dibaca? Jelas tidak jika kita bandingkan dengan versi sinkron. Ini merupakan contoh sederhana dalam penggambaran masalah yang sering dihadapi di kasus nyata ketika bermain dengan proses asinkron. Kami baru mencontohkan penggunaan dua fungsi asinkron yang saling membutuhkan, bagaimana jika ada tiga atau bahkan lebih? Tentu kodenya lebih semrawut untuk dibaca.

Menurut kami, masalah yang berhubungan dengan readability code adalah masalah yang serius. Saking seriusnya, masalah ini dikenal sebagai callback hell atau pyramid of doom. Anda bisa lihat betapa menyeramkannya lengkungan kode ketika kita banyak menggunakan callback, hingga bisa tampak seperti gambar di bawah ini.  
[](https://www.dicoding.com/academies/256/tutorials/15815?from=15810)

Pola callback dapat digunakan pada bahasa pemrograman apa pun, begitu juga masalah yang dihasilkan olehnya. Beruntunglah di JavaScript terdapat cara lain dalam menangani proses asynchronous lebih baik lagi dengan menggunakan Promise. Promise dapat membuat kode kita lebih mudah dibaca dibanding callback. Berikut adalah gambaran dari kode yang memanfaatkan Promise dalam proses asynchronous-nya.

## Asynchronous Handling dengan Promise

Semenjak ECMAScript 2015 (ES6), pola callback bukan menjadi satu-satunya cara dalam menangani proses asynchronous di JavaScript. Di era ES6 atau kita sebut era JavaScript modern, hadir Promise yang menjadi fitur dasar dalam menjalankan operasi asynchronous. Saat ini, banyak sekali fungsi-fungsi di Node.js atau Browser API yang memanfaatkan Promise dibandingkan dengan pola callback dalam menangani proses asynchronous-nya.

Bagi Anda yang pertama kali mendengar Promise mungkin heran dan bertanya mengapa fitur ini dinamakan “Promise” alias “Janji”? Ketahuilah bahwa penamaan ini sebenarnya cukup akurat dan memang Promise bisa dianalogikan sebagai sebuah janji.

Jika kita pikir secara mendalam, sebenarnya proses yang dijalankan secara asynchronous merupakan sebuah janji karena hasil dari proses tersebut tidak bisa langsung kita dapatkan, melainkan kita perlu menunggu beberapa saat. Sama halnya dengan janji di dunia nyata yang butuh waktu untuk terpenuhi.

Selain sama-sama membutuhkan waktu, kesamaan lainnya terjadi pada hasil yang didapatkan. Di dunia nyata, janji bisa terpenuhi dan bisa juga gagal terpenuhi. Contoh, jika teman Anda berjanji akan mengajak lari di sore hari, bisa saja janji tersebut gagal terpenuhi karena hujan atau alasan lainnya. Promise di JavaScript pun memiliki konsep yang sama karena memiliki beberapa kondisi yaitu pending, fulfilled, dan rejected.

* Pending merupakan keadaan Promise sedang berjalan.
* Fulfilled merupakan keadaan Promise yang terpenuhi.
* Rejected merupakan keadaan Promise yang gagal terpenuhi.

Sudah paham mengapa fitur ini dinamakan dengan Promise? Jika sudah, pertanyaan selanjutnya adalah apa bentuk Promise ini di JavaScript? Bagaimana cara menggunakannya?

Promise di JavaScript merupakan sebuah objek yang merepresentasikan operasi asynchronous. Dengan memanfaatkan Promise, fungsi yang berjalan secara asynchronous dapat ditangani dengan abstraksi yang lebih sederhana. Contohnya, alih-alih kita memanfaatkan callback yang dikirim via argumen fungsi, kita bisa mengembalikan mengembalikan objek Promise langsung pada fungsi yang berjalan secara asynchronous.

Jika penjelasannya masih membingungkan, mari kita ulas kembali fungsi getUsers() yang memanfaatkan callback pada materi sebelumnya.

1. function getUsers(isOffline, callback) {
2. // simulate network delay
3. setTimeout(() => {
4. const users = ['John', 'Jack', 'Abigail'];
6. if (isOffline) {
7. callback(new Error('cannot retrieve users due offline'), null);
8. return;
9. }

12. callback(null, users);
13. }, 3000);
14. }

Anda pasti sudah familier dengan kode di atas. Fungsi getUsers() berjalan secara asynchronous dan memanfaatkan argumen callback untuk mengirimkan nilainya.

Jika kita ubah pendekatannya dengan menggunakan Promise, kita tidak membutuhkan lagi argumen callback. Namun, sebagai gantinya fungsi getUsers() perlu mengembalikan objek Promise dengan cara seperti ini.

1. function getUsers(isOffline) {
2. // return a Promise object
3. return new Promise((resolve, reject) => {
5. // simulate network delay
6. setTimeout(() => {
7. const users = ['John', 'Jack', 'Abigail'];
9. if (isOffline) {
10. reject(new Error('cannot retrieve users due offline'));
11. return;
12. }
14. resolve(users);
15. }, 3000);
17. });
18. }

Perhatikan kode di atas. Alih-alih fungsi getUsers() menerima callback sebagai argumen, dia mengembalikan objek Promise. Lalu bagaimana dengan nilai yang sebelum dibawa oleh argumen callback? Nah, di sini Promise menawarkan penggantinya dengan memanfaatkan resolve dan reject. Di kasus yang menyebabkan proses asynchronous error, kita bisa bawa error tersebut menggunakan reject, di kasus proses asynchronous berjalan sukses, kita bisa bawa nilai tersebut dengan resolve. Sudah paham sampai di sini?

**Catatan:** Objek Promise dibuat dengan cara memanggil constructor Promise, yakni new Promise(). Kemudian di dalam constructor, Anda wajib memberikan argumen berupa fungsi yang di dalamnya terdapat operasi asynchronous. Fungsi tersebut juga diberikan akses terhadap dua argumen, yaitu resolve dan reject. Kedua argumen ini bisa Anda manfaatkan dalam membawa hasil dari proses asynchronous berupa data ataupun error.

Sekarang Anda sudah paham cara membuat Promise. Selanjutnya Anda perlu tahu cara menggunakan Promise untuk mendapatkan nilai yang dibawa olehnya. Mari kita lihat cara memanggil dan mendapatkan hasil dari fungsi getUsers().

Setelah fungsi getUsers() diubah dari callback menjadi Promise-based, cara penggunaan fungsinya pun akan berubah. Fungsi yang mengembalikan objek Promise akan memiliki fungsi .then dan .catch yang digunakan untuk mengambil nilai yang dibawa oleh resolve dan reject. Gambarannya seperti ini.

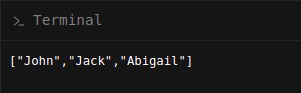
1. getUsers(false)
2. .then(users => console.log(users))
3. .catch(err => console.log(err.message));

Nilai yang dibawa oleh resolve() dapat diambil melalui fungsi .then() dan seketika status Promise akan berubah dari pending ke fulfilled. Sedangkan, nilai yang dibawa oleh reject() dapat diambil melalui fungsi .catch() dan seketika status Promise akan berubah dari pending ke rejected.

Perhatikan juga bahwa .then() dan .catch() selalu mengembalikan Promise, itulah yang menyebabkan pemanggilannya dapat berantai seperti .then().catch().

Mari kita lihat kode lengkapnya agar dapat Anda coba dan jalankan.

Jalankan kode di atas, maka fungsi .then() akan terpanggil dan menampilkan data users pada console.

[](https://www.dicoding.com/academies/256/tutorials/14017?from=15815)

Fungsi .catch() tidak akan terpanggil karena Promise tidak pernah berstatus rejected. Jika Anda ingin Promise menghasilkan status rejected, ubahlah nilai argumen pada pemanggilan getUsers() menjadi true.

[](https://www.dicoding.com/academies/256/tutorials/14017?from=15815)

### Mengubah Callback menjadi Promise dengan Promisify

Di materi sebelumnya kita sudah belajar cara mengubah callback menjadi Promise. Sekarang kita akan coba mengubah callback menjadi Promise dengan pendekatan yang berbeda. Di Node.js, terdapat sebuah fungsi dari core module util yang dapat mengubah fungsi asynchronous callback-based menjadi Promise-based dengan mudah, fungsi tersebut bernama promisify.

Mari kita angkat kembali kode dari fungsi getUsers() callback-based berikut.

1. function getUsers(isOffline, callback) {
2. // simulate network delay
3. setTimeout(() => {
4. const users = ['John', 'Jack', 'Abigail'];
6. if (isOffline) {
7. callback(new Error('cannot retrieve users due offline'), null);
8. return;
9. }

12. callback(null, users);
13. }, 3000);
14. }

Di materi sebelumnya kita sudah mengubahnya menjadi Promise-based dengan me-refactor kode fungsi tersebut. Namun, dalam skenario nyata, sering kali kita tidak memiliki akses untuk me-refactor sebuah fungsi contohnya ketika menggunakan library pihak ketiga. Ketika menghadapi kasus tersebut, bagaimana cara termudah dalam mengubah fungsi tersebut menjadi Promise-based?

Node.js menawarkan solusi yaitu menggunakan fungsi promisify dari core module util. Dengan fungsi tersebut kita bisa secara mudah membuat fungsi baru yang mengimplementasikan Promise berdasarkan fungsi callback-based, contohnya seperti ini.

1. const { promisify } = require('util');
3. function getUsers(isOffline, callback) {
4. // simulate network delay
5. setTimeout(() => {
6. const users = ['John', 'Jack', 'Abigail'];
7. if (isOffline) {
8. callback(new Error('cannot retrieve users due offline'), null);
9. return;
10. }
12. callback(null, users);
13. }, 3000);
14. }
16. // create a Promise version of getUsers
17. const getUsersPromise = promisify(getUsers);

Sekarang, kita memiliki fungsi getUsers(), tetapi menerapkan Promise tanpa perlu mengubahnya secara manual.

1. getUsersPromise(false)
2. .then(users => console.log(users)) // ['John', 'Jack', 'Abigail']
3. .catch(err => console.log(err.message));
5. getUsersPromise(true)
6. .then(users => console.log(users))
7. .catch(err => console.log(err.message)); // cannot retrieve users due offline

Sangat mudah, bukan? Namun, terdapat beberapa catatan ketika Anda hendak mengubah fungsi asynchronous callback-based menjadi Promise-based menggunakan util.promisify().

1. Promisify selalu menganggap callback berada di argumen paling akhir sebuah fungsi asynchronous. Dengan begitu, Anda tidak dapat menggunakan promisify jika callback berada di posisi awal ataupun tengah argumen.
2. Promisify akan bekerja dengan baik jika callback memiliki struktur argumen callback(error, data). Jika callback memiliki struktur argumen di luar aturan tersebut, nilai yang dibawa oleh Promise ketika fulfilled dan rejected bisa tertukar

**Catatan:** Singkatnya, promisify hanya dapat digunakan pada fungsi yang mengikuti aturan Node.js callback. Inilah salah satu alasan perlunya mengikuti aturan yang standar.

Dalam kasus nyata, Anda bisa gunakan promisify dalam mengubah berbagai fungsi yang disediakan Node.js menjadi Promise-based, contohnya fungsi fs.readFile() yang digunakan untuk membaca berkas secara asynchronous.

## Kuis Coding : Mengubah Fungsi Asynchronous Callback-Based menjadi Promise-Based

Untuk menguji kemampuan praktika Anda dalam memahami materi mengubah callback-based menjadi Promise-based, selesaikan kuis berikut.

Sebelum Anda mengerjakan kuis, ada beberapa hal yang perlu Anda perhatikan.

* Mohon untuk membaca secara seksama perintah, kriteria, atau soal pada komentar kode yang diberi tanda TODO.
* Lakukan pengujian pada kode yang Anda tulis contohnya dengan menggunakan console.log() untuk memastikan hasil sesuai dengan yang Anda harapkan. Untuk menjalankan kode, gunakan tombol **Jalankan**.

Bila sudah yakin dengan pekerjaan yang Anda lakukan, submit kode dengan klik tombol Submit kode. Jika kode yang di-submit benar, Anda bisa melanjutkan ke modul berikutnya. Jika tidak, Anda bisa memperbaiki kembali kode sebanyak yang Anda mau.

Batas waktu tunggu mengirimkan kembali perbaikan kode adalah 1 menit. Jadi pastikan kode yang Anda tulis sudah memenuhi perintah, kriteria, atau soal pada kuis kali ini.

Good luck!

|  |
| --- |
| **Mengalami kendala?** Cobalah kunjungi [Forum Diskusi](https://www.dicoding.com/academies/256/discussions?tutorial=14027) untuk mencari solusi atau bertanya langsung kepada mentor kami. |

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/14027#run1-editor1)



1

/\*\*

2

\* @TODO

3

\* Ubahlah fungsi asynchronous callback-based getProvinces menjadi Promise-based.

4

\*

5

\* Catatan:

6

\* - Anda boleh menggunakan util.promisify untuk mengubah fungsi callback-based menjadi Promise-based.

7

\* - Jika nama fungsinya berubah, sesuaikan nilai yang diekspor tanpa mengubah nama properti di akhir berkas ini.

8

\*/

9

​

10

​

11

function getProvinces(countryId, callback) {

12

setTimeout(() => {

13

if (countryId === 'id') {

14

callback(null, [

15

{ id: 'id-jk', name: 'Jakarta' },

16

{ id: 'id-bt', name: 'Banten' },

17

{ id: 'id-jr', name: 'Jawa Barat' },

18

]);

19

return;

20

}

 Input  Reset

 Jalankan  Submit Kode

## Promise Berantai

Masih ingat tentang masalah readability code yang disebabkan callback hell? Masalah tersebut kerap terjadi ketika proses agregasi data dari berbagai proses asynchronous yang saling bergantung antar satu dengan yang lainnya. Dengan hadirnya Promise, kita bisa terbebas dari callback hell.

Promise didesain untuk dapat dijalankan secara berantai dengan Promise lain. Contohnya, jika Anda memiliki dua fungsi Promise, Anda bisa menjalankan kedua fungsi tersebut secara terurut, alias berantai. Agar penjelasannya lebih seru, mari kita banyangkan dengan kehidupan sehari-hari.

Dalam kehidupan sehari-hari, kita mungkin pernah melakukan chaining Promise tanpa disadari. Misalkan ketika ingin menonton film di bioskop, kita perlu melakukan beberapa tahap sebelum akhirnya bisa menikmati film tersebut. Misalnya, tahap pertama kita perlu **menarik uang** untuk nantinya membeli tiket bioskop. Tahap ini bisa direpresentasikan menjadi sebuah fungsi bernama withdrawMoney(). Fungsi tersebut akan mengembalikan sejumlah uang jika saldonya mencukupi, dalam arti lain Promise-nya menjadi fulfilled (resolve). Selain itu, fungsi akan menampilkan pesan error jika saldonya kurang yang artinya rejected (reject). Jika Promise-nya fulfilled, tahap selanjutnya akan dijalankan.

Tahap kedua adalah **membeli tiket bioskop** yang dapat direpresentasikan dengan fungsi bernama buyCinemaTicket(). Fungsi tersebut menerima argumen uang yang sudah kita tarik. Jika uang yang diberikan pada argumen kurang dari harga tiketnya, Promise akan di-reject dengan pesan “not enough money to buy ticket”. Jika nilai argumen cukup, Promise akan mengembalikan nilai “tiket-1”.

Tahap ketiga adalah **masuk ke bioskop**, aktivitas ini bisa digambarkan dengan fungsi goInsideCinema() yang menerima satu argumen berupa tiket. Jika tidak ada tiket, Promise akan di-reject dengan pesan “no ticket”. Jika ada, Promise akan di-resolve dengan pesan “enjoy the movie”.

Seluruh tahap yang kami jelaskan adalah Promise dan dapat dijalankan secara berantai seperti ini.

Ketika kita memanggil fungsi watchMovie(), Promise chaining akan dijalankan. Dengan teknik tersebut, kita bisa menentukan urutan pemanggilan tiap-tiap fungsi Promise berdasarkan tahapan yang harus dilakukan. Pada kode di atas, kita bisa melihat bahwa tahapan pertama adalah memanggil fungsi withDrawMoney(), kemudian buyCinemaTikcet(), dan terakhir goInsideCinema(). Setelah tahapan tersebut selesai, nilai akhir dari Promise akan ditampilkan ke dalam console dengan fungsi console.log(). Jika ada Promise yang di-reject pada salah satu tahapan, ia akan langsung ditangkap oleh catch dan menampilkan error ke console dengan fungsi console.log().

Dalam chaining Promise seperti yang ditunjukkan oleh kode di atas, penting untuk diingat bahwa setiap fungsi .then() harus mengembalikan Promise baru untuk melanjutkan proses chaining karena nilai yang dikembalikan tersebut akan dibawa ke dalam fungsi .then() selanjutnya.

Ada tips dalam melakukan Promise chaining di atas. Anda bisa membuat kode tampak lebih bersih dan singkat dengan memanfaatkan arrow function implisit return. Namun, tips ini hanya dapat digunakan jika arrow function hanya memiliki satu baris kode saja.

1. function watchMovie() {
2. withDrawMoney(10)
3. .then((money) => buyCinemaTicket(money))
4. .then((ticket) => goInsideCinema(ticket))
5. .then((result) => console.log(result))
6. .catch((error) => console.log(error.message));
7. }

## Promise Static Method

Dalam JavaScript, Promise merupakan salah satu fitur yang penting terutama ketika bekerja dengan operasi asynchronous. Di materi sebelumnya, Anda sudah mengenal dasar-dasar dari penggunaan Promise.

Promise memiliki beberapa static method yang dapat digunakan untuk mempermudah penggunaan di berbagai kasus. Di materi kali ini, kami akan membahas empat static method dari class Promise yang umum digunakan, yaitu Promise.all, Promise.race, Promise.allSettled, dan Promise.any.

#### Promise.all

Promise.all() digunakan untuk mengeksekusi banyak Promise secara paralel. Method ini menerima sebuah array yang berisi beberapa Promise sebagai argumen. Fungsi ini akan mengembalikan sebuah Promise baru dan membawa nilai yang di-resolve pada masing-masing Promise yang diletakan pada array argumen.

Berikut adalah contoh penggunaan Promise.all().

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/14042?from=14037#run1-editor1)



1

const promise1 = new Promise((resolve) => setTimeout(() => resolve(1), 1000));

2

const promise2 = new Promise((resolve) => setTimeout(() => resolve(2), 2000));

3

const promise3 = new Promise((resolve) => setTimeout(() => resolve(3), 3000));

4

​

5

Promise.all([promise1, promise2, promise3]).then((values) => console.log(values)); // [1, 2, 3] setelah 3 detik

 Input  Reset

 Jalankan

Pada contoh di atas, kita membuat tiga Promise yang masing-masing akan mengembalikan setelah waktu yang berbeda-beda. Kemudian, kita menggunakan Promise.all() untuk mengeksekusi ketiga Promise tersebut secara paralel. Ketika semua Promise selesai dieksekusi, Promise.all() akan mengembalikan Promise baru yang membawa nilai array yang berisi nilai-nilai yang dikembalikan oleh ketiga Promise tersebut.

Jika terdapat rejection pada salah satu Promise yang ada, Promise.all() akan langsung menghasilkan rejected tanpa mengembalikan nilai Promise lain yang statusnya fulfilled. Berikut contohnya.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/14042?from=14037#run2-editor1)



1

const promise1 = new Promise((resolve) => setTimeout(() => resolve(1), 1000));

2

const promise2 = new Promise((resolve, reject) => setTimeout(() => reject(new Error('ups')), 2000));

3

const promise3 = new Promise((resolve) => setTimeout(() => resolve(3), 3000));

4

​

5

Promise.all([promise1, promise2, promise3])

6

.then((values) => console.log(values))

7

.catch((error) => console.log(error.message)); // ups

 Input  Reset

 Jalankan

#### Promise.race

Promise.race() digunakan untuk mengeksekusi beberapa Promise secara paralel seperti Promise.all(), tetapi hanya mengembalikan nilai Promise yang paling cepat menyelesaikan eksekusinya.

Berikut contoh penggunaan dar Promise.race().

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/14042?from=14037#run3-editor1)



1

const promise1 = new Promise((resolve) => setTimeout(() => resolve(1), 1000));

2

const promise2 = new Promise((resolve) => setTimeout(() => resolve(2), 2000));

3

const promise3 = new Promise((resolve) => setTimeout(() => resolve(3), 3000));

4

​

5

Promise.race([promise1, promise2, promise3])

6

.then((value) => console.log(value)); // 1 setelah 1 detik

 Input  Reset

 Jalankan

Pada contoh kode di atas, kita membuat tiga Promise yang masing-masing membutuhkan waktu selesai yang berbeda-beda. Kemudian, kita menggunakan Promise.race() untuk mengeksekusi ketiganya secara paralel. Ingat! Promise.race() hanya mengembalikan nilai Promise yang prosesnya paling cepat, maka ia hanya mengembalikan nilai dari promise1.

Lalu bagaimana jika salah satu Promise ada yang rejected? Jika proses rejection merupakan Promise tercepat, ia akan mengembalikan rejection tersebut. Jika rejection tidak kalah cepat dengan Promise lain yang nilainya fulfilled, ia akan tetap mengembalikan nilai resolved tercepat.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/14042?from=14037#run4-editor1)



1

const promise1 = new Promise((resolve, reject) => setTimeout(() => reject(new Error('ups')), 1000));

2

const promise2 = new Promise((resolve) => setTimeout(() => resolve(2), 2000));

3

const promise3 = new Promise((resolve) => setTimeout(() => resolve(3), 3000));

4

​

5

Promise.race([promise1, promise2, promise3])

6

.then((value) => console.log(value))

7

.catch((error) => console.log(error.message)); // Ups

 Input  Reset

 Jalankan

#### Promise.allSettled

Sama seperti yang kami jelaskan sebelumnya, Promise.allSettled() digunakan untuk mengeksekusi beberapa Promise secara paralel. Namun, Promise.allSettled() mengembalikan seluruh hasil dari Promise yang dieksekusi dalam bentuk array of object dengan format berikut ini.

1. {
2. status: 'fulfilled' | 'rejected',
3. value: 'value of the Promise',
4. reason: 'error of the Promise',
5. }

Dari struktur objek yang dihasilkan, Anda bisa melihat bahwa Promise.allSettled() akan mengembalikan seluruh nilai Promise yang dijalankan, baik yang statusnya fulfilled ataupun rejected.

Berikut contoh penggunaan Promise.allSettled().

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/14042?from=14037#run5-editor1)



1

const promise1 = new Promise((resolve) => setTimeout(() => resolve(1), 1000));

2

const promise2 = new Promise((resolve, reject) => setTimeout(() => reject(new Error("Error")), 2000));

3

const promise3 = new Promise((resolve) => setTimeout(() => resolve(3), 3000));

4

​

5

Promise.allSettled([promise1, promise2, promise3])

6

.then((results) => console.log(results));

7

// [{status: "fulfilled", value: 1}, {status: "rejected", reason: Error}, {status: "fulfilled", value: 3}] setelah 3 detik

 Input  Reset

 Jalankan

Pada contoh di atas, kita membuat tiga Promise yang masing-masing membutuhkan waktu eksekusi yang berbeda. Kita menggunakan Promise.allSettled() untuk menjalankan seluruh Promise secara paralel. Pada output console, Anda bisa melihat bahwa Promise.allSettled() mengembalikan array of object seperti yang sudah dijelaskan sebelumnya.

#### Promise.any

Method Promise yang terakhir akan kami bahas adalah Promise.any(). Method ini bekerja mirip seperti Promise.race(), tetapi hanya **mengembalikan nilai tercepat yang berstatus fulfilled**. Jika seluruh Promise berstatus rejected, method ini akan mengembalikan rejected dengan pesan “All Promises were rejected”.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/14042?from=14037#run6-editor1)



1

// fulfilled sample

2

const promiseResolve1 = new Promise((resolve, reject) => setTimeout(() => resolve("1"), 1000));

3

const promiseResolve2 = new Promise((resolve, reject) => setTimeout(() => resolve("2"), 2000));

4

const promiseResolve3 = new Promise((resolve, reject) => setTimeout(() => resolve("3"), 3000));

5

​

6

Promise.any([promiseResolve1, promiseResolve2, promiseResolve3])

7

.then((value) => console.log(value)) // 1

8

.catch((error) => console.log(error.message));

9

​

10

// rejected sample

11

const promiseReject1 = new Promise((resolve, reject) => setTimeout(() => reject(new Error('1')), 1000));

12

const promiseReject2 = new Promise((resolve, reject) => setTimeout(() => reject(new Error('2')), 2000));

13

const promiseReject3 = new Promise((resolve, reject) => setTimeout(() => reject(new Error("3")), 3000));

14

​

15

Promise.any([promiseReject1, promiseReject2, promiseReject3])

16

.then((value) => console.log(value))

17

.catch((error) => console.log(error.message)); // All Promises were rejected

 Input  Reset

 Jalankan

Static method Promise seperti Promise.all(), Promise.race(), Promise.allSettled(), dan Promise.any() dapat sangat membantu ketika bekerja dengan operasi asynchronous menggunakan Promise. Dengan menggunakan method-method tersebut, kita dapat mengeksekusi banyak Promise secara paralel dan mengelola hasilnya dengan lebih efisien.

## Asynchronous Handling dengan Sintaks Async dan Await

Di materi sebelumnya Anda telah belajar banyak mengenai Promise. Kita sepakat bahwa penanganan asynchronous dengan Promise jauh lebih baik daripada menggunakan callback. Dengan menggunakan Promise, kita bisa menulis kode lebih simpel dan mudah dibaca.

Sekarang mari kita bayangkan, bagaimana jika kita bisa menangani proses asynchronous layaknya proses synchronous? Tentunya ini akan membuat kode kita jauh lebih ringkas dan mudah dipahami karena tidak perlu menggunakan .then() dan .catch() untuk mendapatkan nilainya. Untunglah harapan tersebut kini sudah terealisasi dengan hadirnya fitur async dan await di JavaScript versi ECMAScript 2017.

Sintaks async dan await memungkinkan kita untuk menulis kode dalam menangani proses asynchronous Promise-based dengan cara yang sama seperti kode synchronous. Dalam penanganan error-nya (rejection) pun kita bisa menggunakan try dan catch layaknya error yang dihasilkan oleh proses synchronous. Mari kita lihat contohnya dengan mengangkat kembali kasus menonton film di bioskop secara asynchronous.

1. function watchMovie() {
2. withDrawMoney(10)
3. .then((money) => buyCinemaTicket(money))
4. .then((ticket) => goInsideCinema(ticket))
5. .then((result) => console.log(result))
6. .catch((error) => console.log(error.message));
7. }
8. watchMovie();

Kami yakin Anda sudah paham dengan penanganan Promise berantai seperti di atas. Sekarang, bagaimana jika kita memanfaatkan async dan await dalam menangani proses asynchronous-nya? Kodenya akan tampak seperti ini.

1. async function watchMovie() {
2. try {
3. const money = await withDrawMoney(10);
4. const ticket = await buyCinemaTicket(money);
5. const result = await goInsideCinema(ticket);
7. console.log(result);
8. } catch (error) {
9. console.log(error.message);
10. }
11. }
13. watchMovie();

Untuk menggunakan fitur async dan await, kita perlu mengubah sebuah fungsi agar berjalan secara asynchronous dengan menambahkan kata kunci async di awal fungsi. Setelah itu, di dalam fungsi tersebut, kita bisa mengambil data yang dihasilkan oleh Promise dengan menggunakan sintaks await. Sintaks await akan memberitahu JavaScript untuk menunggu proses Promise selesai sebelum mengeksekusi kode baris selanjutnya, sehingga kita bisa melakukan Promise berantai tanpa perlu menggunakan .then(). Simak juga cara penanganan erronya. Kita sudah tidak lagi menggunakan .catch() melainkan try dan catch layaknya proses synchronous.

Ketahuilah bahwa setiap fungsi yang didefinisikan dengan kata async akan selalu mengembalikan sebuah Promise. Contoh pada kode di atas, kita bisa menggunakan fungsi .then() setelah pemanggilan fungsi watchMovie() seperti ini.

1. async function watchMovie() {
2. try {
3. const money = await withDrawMoney(10);
4. const ticket = await buyCinemaTicket(money);
5. const result = await goInsideCinema(ticket);
7. console.log(result);
8. } catch (error) {
9. console.log(error.message);
10. }
11. }
13. watchMovie().then(() => console.log('done'));
15. /\*\* output \*/
16. // enjoy the movie
17. // done

Jika fungsi yang diberikan kata async mengembalikan Promise, bagaimana caranya mengontrol status Promise tersebut agar bernilai fulfilled atau rejected? Kata async secara implisit menggunakan kata return untuk mengubah status Promise menjadi fulfilled dan menggunakan kata throw untuk mengubah status Promise menjadi rejected. Mari kita buktikan dengan mengubah fungsi watchMovie() dengan mengembalikan nilai dan men-throw error.

Lihatlah perubahan kode di atas. Fungsi watchMovie() tidak lagi menampilkan nilainya pada console.log() melainkan mengembalikan nilai result dan bisa juga membangkitkan error. Kemudian dalam memanggil fungsinya, kita bisa .then() dan .catch() untuk mengambil nilai yang di-resolve ataupun di-reject. Hal ini membuktikan bahwa kata async membuat fungsi mengembalikan Promise secara implisit.

Sintaks async dan await memiliki beberapa keuntungan dibandingkan dengan penggunaan callback dan Promise secara langsung:

1. **Lebih mudah dipahami dan ditulis**Sintaks async dan await memungkinkan kita untuk menulis kode yang menangani asynchronous dengan cara yang sama seperti kode sinkronous. Tentu ini membuat kode lebih dipahami dan ditulis.
2. **Terhindar dari callback**Penggunaan Promise secara langsung pun tidak terlepas dari callback seperti .then() dan .catch(). Namun, dengan async dan await, kita bisa terlepas dari pola callback.
3. **Lebih mudah dalam mengelola error**  
   Async dan await benar-benar membuat pengalaman asynchronous layaknya synchronous, hingga dalam penanganan error-nya pun menggunakan beberapa hal yang sudah ada seperti try, catch, dan throw.

## Kuis Coding : Asynchronous Proses secara Berantai

Untuk menguji kemampuan praktikal Anda dalam memahami materi Concurrency di JavaScript, silakan selesaikan kuis berikut.

Sebelum Anda mengerjakan kuis, ada beberapa hal yang perlu Anda perhatikan.

* Mohon untuk membaca secara seksama perintah, kriteria, atau soal pada komentar kode yang diberi tanda **TODO.**
* Lakukan pengujian pada kode yang Anda tulis contohnya dengan menggunakan console.log() untuk memastikan hasil sesuai dengan yang Anda harapkan. Untuk menjalankan kode, gunakan tombol Jalankan.

Bila sudah yakin dengan pekerjaan yang Anda lakukan, submit kode dengan klik tombol **Submit kode.** Jika kode yang di-submit benar, Anda bisa melanjutkan ke modul berikutnya. Jika tidak, Anda bisa memperbaiki kembali kode sebanyak yang Anda mau.

Batas waktu tunggu untuk mengirimkan kembali perbaikan kode adalah 1 menit. Jadi pastikan kode yang Anda tulis sudah memenuhi perintah, kriteria, atau soal pada kuis kali ini.

Good luck!

|  |
| --- |
| **Mengalami kendala?** Cobalah kunjungi [Forum Diskusi](https://www.dicoding.com/academies/256/discussions?query=&query_criteria=&sort=&sort_direction=&title=&tutorial=15930&keywords=&creator=&page=1) untuk mencari solusi atau bertanya langsung kepada mentor kami. |

Selamat, Anda telah menyelesaikan modul ini.  
Kode yang Anda kirimkan telah berfungsi dengan baik.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/15930#run1-editor1)
* [utils.js](https://www.dicoding.com/academies/256/tutorials/15930#run1-editor2)



47

​

48

};

49

​

50

​

51

// TODO: 2

52

async function gettingUserName(){

53

try{

54

const user = await fetchingUserFromInternet(false);

55

return user.name;

56

}catch(error){

57

return error.message;

58

}

59

};

60

​

61

gettingUserName().then((value)=>{

62

console.log(value);

63

});

64

​

65

/\*\*

66

\* Abaikan kode di bawah ini

67

\*/

68

​

69

module.exports = { fetchingUserFromInternet, gettingUserName, NetworkError };

70

​

 Input  Reset

 Jalankan  Submit Kode

## Rangkuman Asynchronous Process

### Pengenalan Asynchronous

Dalam asynchronous process, kita bisa menjalankan proses yang berada di baris kedua tanpa harus menunggu kode pada baris pertama selesai dieksekusi. Dalam arti lain, kita bisa melakukan lebih dari satu proses sekaligus dalam waktu yang bersamaan.

Umumnya, kita memanfaatkan asynchronous pada proses yang besar dan membutuhkan waktu lama, seperti mengambil data dari internet, menyimpan data ke database, dan membaca data dari sebuah berkas.

### setTimeout Function

JavaScript merupakan bahasa pemrograman yang menerapkan pola event-driven, baik pada environment Node.js maupun browser. Seluruh proses yang berjalan dipicu berdasarkan sebuah event atau kejadian, contohnya [DOMContentLoaded](https://developer.mozilla.org/en-US/docs/Web/API/Window/DOMContentLoaded_event) pada browser atau process exit pada Node.js. Dengan pola event-driven itu, artinya kode yang ditulis dengan JavaScript tidak harus dijalankan secara berurutan, tetapi kode dapat dijalankan berdasarkan event atau bahkan dijadwalkan.

Ada satu fungsi dalami JavaScript yang digunakan untuk menjadwalkan pemanggilan sebuah kode sekaligus membuatnya  berjalan secara asynchronous, yakni setTimeout(). Fungsi tersebut menerima dua argumen dengan penjelasan berikut.

1. Argumen pertama merupakan sebuah fungsi yang akan dipanggil secara terjadwal dan asynchronous.
2. Argumen kedua merupakan delay waktu dalam satuan milisecond yang menentukan delay dari pemanggilan fungsi padai argumen pertama.

### Asynchronous Handling dengan Callback

Pola callback merupakan salah satu cara menangani proses asynchronous. Tidak hanya di JavaScript, pola ini juga digunakan pada bahasa pemrograman lainnya. Callback merupakan sebuah fungsi yang disisipkan pada argumen fungsi asynchronous dan akan dipanggil ketika proses dinyatakan selesai. Fungsi callback akan membawa nilai-nilai yang dihasilkan dari proses asynchronous sehingga kita bisa memanfaatkan nilai tersebut.

Berikut adalah contoh dari penerapan callback.

1. function getUsers(callback) {
2. // simulate network delay
3. setTimeout(() => {
4. const users = ['John', 'Jack', 'Abigail'];
6. callback(users);
7. }, 3000);
8. }
10. function usersCallback(users) {
11. console.log(users);
12. }
14. getUsers(usersCallback);

#### Callback Hell

Seiring kompleksnya pengembangan aplikasi, kita akan semakin banyak menggunakan fungsi yang berjalan secara asynchronous. Tidak jarang kita perlu mengagregasikan banyak data dari berbagai proses asynchronous. Masalah akan timbul ketika sebuah proses asynchronous perlu dijalankan setelah proses asynchronous lainnya selesai.

Berikut adalah contoh kode proses asynchronous dengan callback yang saling tergantung satu sama lainnya.

1. function getUserWeather(userId, callback) {
2. getUser(userId, (error, user) => {
3. if (error) {
4. callback(error, null);
5. }
7. getWeather(user.location, (error, weather) => {
8. if (error) {
9. callback(error, null);
10. }
12. callback(null, { ...user, ...weather });
13. });
14. });
15. }
17. getUserWeather(1, (error, userWeather) => {
18. if (error) {
19. console.log(error.message);
20. }
22. console.log(userWeather); // { id: 1, name: 'John Doe', location: 'Jakarta', weather: 'Sunny', temperature: 30 }
23. });

Ini merupakan contoh sederhana dalam penggambaran masalah yang sering dihadapi di kasus nyata ketika bermain dengan proses asinkron. Menurut kami, masalah yang berhubungan dengan readability code adalah masalah yang serius. Saking seriusnya, masalah ini dikenal sebagai callback hell atau pyramid of doom.

Pola callback dapat digunakan pada bahasa pemrograman apa pun, begitu juga masalah yang dihasilkan olehnya. Beruntunglah di JavaScript terdapat cara lain dalam menangani proses asynchronous lebih baik lagi dengan menggunakan Promise. Promise dapat membuat kode kita lebih mudah dibaca dibanding callback. Berikut adalah gambaran dari kode yang memanfaatkan Promise dalam proses asynchronous-nya.

1. const { getUser, getWeather } = require('./utils');
3. function getUserWeather(userId) {
4. let user;
6. return getUser(userId)
7. .then((\_user) => {
8. user = \_user;
9. return getWeather(user.location)
10. })
11. .then((weather) => ({ ...user, ...weather }))
12. }
14. getUserWeather(1)
15. .then(console.log)
16. .catch(console.log);

### Asynchronous Handling dengan Promise

Semenjak ECMAScript 2015 (ES6), pola callback bukan menjadi satu-satunya cara dalam menangani proses asynchronous di JavaScript. Di era ES6 atau kita sebut era JavaScript modern, hadir Promise yang menjadi fitur dasar dalam menjalankan operasi asynchronous. Saat ini, banyak sekali fungsi-fungsi di Node.js atau Browser API yang memanfaatkan Promise dibandingkan dengan pola callback dalam menangani proses asynchronous-nya.

Bagi Anda yang pertama kali mendengar Promise mungkin heran dan bertanya mengapa fitur ini dinamakan “Promise” alias “Janji”? Ketahuilah bahwa penamaan ini sebenarnya cukup akurat dan memang Promise bisa dianalogikan sebagai sebuah janji.

Jika kita pikir secara mendalam, sebenarnya proses yang dijalankan secara asynchronous merupakan sebuah janji karena hasil dari proses tersebut tidak bisa langsung kita dapatkan, melainkan kita perlu menunggu beberapa saat. Sama halnya dengan janji di dunia nyata yang butuh waktu untuk terpenuhi.

Selain sama-sama membutuhkan waktu, kesamaan lainnya terjadi pada hasil yang didapatkan. Di dunia nyata, janji bisa terpenuhi dan bisa juga gagal terpenuhi. Contoh, jika teman Anda berjanji akan mengajak lari padai sore hari, bisa saja janji tersebut gagal terpenuhi karena hujan atau alasan lainnya. Promise di JavaScript pun memiliki konsep yang sama karena memiliki beberapa kondisi yaitu pending, fulfilled, dan rejected.

* Pending merupakan keadaan Promise sedang berjalan.
* Fulfilled merupakan keadaan Promise yang terpenuhi.
* Rejected merupakan keadaan Promise yang gagal terpenuhi.

Berikut adalah contoh fungsi asynchronous yang memanfaatkan Promise.

1. function getUsers(isOffline) {
2. // return a Promise object
3. return new Promise((resolve, reject) => {
5. // simulate network delay
6. setTimeout(() => {
7. const users = ['John', 'Jack', 'Abigail'];
9. if (isOffline) {
10. reject(new Error('cannot retrieve users due offline'));
11. return;
12. }
14. resolve(users);
15. }, 3000);
17. });
18. }

Fungsi yang mengembalikan objek Promise akan memiliki fungsi .then dan .catch yang digunakan untuk mengambil nilai yang dibawa oleh resolve dan reject. Gambarannya seperti ini.

1. getUsers(false)
2. .then(users => console.log(users))
3. .catch(err => console.log(err.message));

#### Mengubah Callback menjadi Promise dengan Promisify

Node.js menawarkan solusi mudah dalam mengubah fungsi asynchronous callback-based menjadi Promise-based, yaitu menggunakan fungsi promisify dari core module util. Dengan fungsi tersebut kita bisa secara mudah membuat fungsi baru yang mengimplementasikan Promise berdasarkan fungsi callback-based, contohnya seperti ini.

1. const { promisify } = require('util');
3. function getUsers(isOffline, callback) {
4. // simulate network delay
5. setTimeout(() => {
6. const users = ['John', 'Jack', 'Abigail'];
7. if (isOffline) {
8. callback(new Error('cannot retrieve users due offline'), null);
9. return;
10. }
12. callback(null, users);
13. }, 3000);
14. }
16. // create a Promise version of getUsers
17. const getUsersPromise = promisify(getUsers);

Sekarang, kita memiliki fungsi getUsers(), tetapi menerapkan Promise tanpa perlu mengubahnya secara manual.

1. getUsersPromise(false)
2. .then(users => console.log(users)) // ['John', 'Jack', 'Abigail']
3. .catch(err => console.log(err.message));
5. getUsersPromise(true)
6. .then(users => console.log(users))
7. .catch(err => console.log(err.message)); // cannot retrieve users due offline

Sangat mudah, bukan? Namun, terdapat beberapa catatan ketika Anda hendak mengubah fungsi asynchronous callback-based menjadi Promise-based menggunakan util.promisify().

1. Promisify selalu menganggap callback berada padai argumen paling akhir sebuah fungsi asynchronous. Dengan begitu, Anda tidak dapat menggunakan promisify jika callback berada di posisi awal ataupun tengah argumen.
2. Promisify akan bekerja dengan baik jika callback memiliki struktur argumen callback(error, data). Jika callback memiliki struktur argumen di luar aturan tersebut, nilai yang dibawa oleh Promise ketika fulfilled dan rejected bisa tertukar.

#### Promise Berantai

Promise didesain untuk dapat dijalankan secara berantai dengan Promise lain. Contohnya, jika Anda memiliki dua fungsi Promise, Anda bisa menjalankan kedua fungsi tersebut secara berurut, alias berantai. Contohnya bisa Anda lihat pada kode di bawah ini.

1. function withDrawMoney(amount) {
2. return new Promise((resolve, reject) => {
3. setTimeout(() => {
4. if (amount > 100) {
5. reject(new Error('Not enough money to withdraw'))
6. }
8. resolve(amount)
9. }, 1000);
10. });
11. }
13. function buyCinemaTicket(money) {
14. return new Promise((resolve, reject) => {
15. setTimeout(() => {
16. if (money < 10) {
17. reject(new Error('not enough money to buy ticket'))
18. }
20. resolve('ticket-1')
21. }, 1000);
22. });
23. }
25. function goInsideCinema(ticket) {
26. return new Promise((resolve, reject) => {
27. setTimeout(() => {
28. if (!ticket) {
29. reject(new Error('no ticket'))
30. }

33. resolve('enjoy the movie')
34. }, 1000);
35. });
36. }
38. function watchMovie() {
39. withDrawMoney(10)
40. .then((money) => {
41. return buyCinemaTicket(money)
42. })
43. .then((ticket) => {
44. return goInsideCinema(ticket)
45. })
46. .then((result) => {
47. console.log(result)
48. })
49. .catch((error) => {
50. console.log(error.message)
51. });
52. }
54. watchMovie();

Ketika kita memanggil fungsi watchMovie(), Promise chaining akan dijalankan. Dengan teknik tersebut, kita bisa menentukan urutan pemanggilan tiap-tiap fungsi Promise berdasarkan tahapan yang harus dilakukan. Pada kode di atas, kita bisa melihat bahwa tahapan pertama adalah memanggil fungsi withDrawMoney(), kemudian buyCinemaTikcet(), dan terakhir goInsideCinema(). Setelah tahapan tersebut selesai, nilai akhir dari Promise akan ditampilkan ke dalam console dengan fungsi console.log(). Jika ada Promise yang di-reject pada salah satu tahapan, ia akan langsung ditangkap oleh catch dan menampilkan error ke console dengan fungsi console.log().

Dalam chaining Promise seperti yang ditunjukkan oleh kode di atas, penting untuk diingat bahwa setiap fungsi .then() harus mengembalikan Promise baru untuk melanjutkan proses chaining karena nilai yang dikembalikan tersebut akan dibawa ke dalam fungsi .then() selanjutnya.

### Promise Static Method

Promise memiliki beberapa static method yang dapat digunakan untuk mempermudah penggunaan di berbagai kasus. Dalami materi kali ini, kami akan membahas empat static method dari class Promise yang umum digunakan, yaitu Promise.all, Promise.race, Promise.allSettled, dan Promise.any.

#### Promise.all

Promise.all() digunakan untuk mengeksekusi banyak Promise secara paralel. Method ini menerima sebuah array yang berisi beberapa Promise sebagai argumen. Fungsi ini akan mengembalikan sebuah Promise baru dan membawa nilai yang di-resolve pada masing-masing Promise yang diletakkan pada array argumen.

#### Promise.race

Promise.race() digunakan untuk mengeksekusi beberapa Promise secara paralel seperti Promise.all(), tetapi hanya mengembalikan nilai Promise yang paling cepat menyelesaikan eksekusinya.

#### Promise.allSettled

Sebagaimana seperti yang kami jelaskan sebelumnya, Promise.allSettled() digunakan untuk mengeksekusi beberapa Promise secara paralel. Namun, Promise.allSettled() mengembalikan seluruh hasil dari Promise yang dieksekusi dalam bentuk array of object dengan format berikut ini.

1. {
2. status: 'fulfilled' | 'rejected',
3. value: 'value of the Promise' | 'error of the Promise',
4. }

#### Promise.any

Method Promise yang terakhir akan kami bahas adalah Promise.any(). Method ini bekerja mirip seperti Promise.race(), tetapi **hanya mengembalikan nilai tercepat yang berstatus fulfilled**. Jika seluruh Promise berstatus rejected, method ini akan mengembalikan rejected dengan pesan “All Promises were rejected”.

### Asynchronous Handling dengan Async dan Await

Sintaks async dan await memungkinkan kita untuk menulis kode dalam menangani proses asynchronous Promise-based dengan cara yang sama seperti kode synchronous. Dalam penanganan error-nya (rejection) pun kita bisa menggunakan try dan catch layaknya error yang dihasilkan oleh proses synchronous.

1. async function watchMovie() {
2. try {
3. const money = await withDrawMoney(10);
4. const ticket = await buyCinemaTicket(money);
5. const result = await goInsideCinema(ticket);
7. console.log(result);
8. } catch (error) {
9. console.log(error.message);
10. }
11. }
13. watchMovie();

Untuk menggunakan fitur async dan await, kita perlu mengubah sebuah fungsi agar berjalan secara asynchronous dengan menambahkan kata kunci async di awal fungsi. Setelah itu, di dalam fungsi tersebut, kita bisa mengambil data yang dihasilkan oleh Promise dengan menggunakan sintaks await. Sintaks await akan memberitahu JavaScript untuk menunggu proses Promise selesai sebelum mengeksekusi kode baris selanjutnya. Jadi, kita bisa melakukan Promise berantai tanpa perlu menggunakan .then(). Simak juga cara penanganan errornya. Kita sudah tidak lagi menggunakan .catch(), melainkan try dan catch layaknya proses synchronous.

Aturan

Knowledge Check : Asynchronous Process bertujuan untuk menguji pengetahuan Anda tentang materi yang telah dipelajari di modul ini.

Terdapat 2 pertanyaan yang harus dikerjakan dalam kuis ini. Beberapa ketentuannya sebagai berikut:

* Syarat nilai kelulusan : 100%
* Durasi ujian : 5 menit

Apabila tidak memenuhi syarat kelulusan, Anda bisa mengulang pengerjaan kuis kembali.

Selamat Mengerjakan!

# [Node Package Manager](https://www.dicoding.com/academies/256/tutorials/14057)

## [Pengenalan Node Package Manager](https://www.dicoding.com/academies/256/tutorials/14057)

[Dalam pengembangan aplikasi, kita tidak akan lepas dari package atau library (selanjutnya akan disebut package). Package di sini merupakan sebuah kode yang dibuat untuk menyelesaikan suatu masalah. Contohnya ketika aplikasi yang kita buat membutuhkan fitur kalender sementara fitur tersebut tidak didukung secara default oleh JavaScript. Alih-alih membuat fitur kalender dari nol, kita dapat menggunakan package yang telah dibuat oleh developer lain. Waktu pembuatan fitur menjadi lebih singkat!](https://www.dicoding.com/academies/256/tutorials/14057)

[Pada modul ini kita akan belajar mengenai beberapa hal berikut:](https://www.dicoding.com/academies/256/tutorials/14057)

* [Menambahkan package ke dalam project JavaScript.](https://www.dicoding.com/academies/256/tutorials/14057)
* [Menggunakan package untuk membantu pengembangan aplikasi.](https://www.dicoding.com/academies/256/tutorials/14057)
* [Menghapus package yang sudah tidak digunakan.](https://www.dicoding.com/academies/256/tutorials/14057)

[Mari kita mulai!](https://www.dicoding.com/academies/256/tutorials/14057)

## Getting Started with NPM

Semakin kompleks suatu aplikasi, semakin banyak pula package yang digunakan. Di sinilah kita memerlukan sebuah package manager. Package manager merupakan tools yang dapat membantu pengelolaan package pada proyek agar lebih mudah. Kita dapat dengan mudah memasang dan menghapus package yang dibuat oleh orang lain pada proyek kita. Selain itu, kita juga dapat menulis module/package untuk dikonsumsi secara publik.

Hampir setiap bahasa pemrograman populer memiliki package manager-nya masing-masing. Java memiliki Maven dan Gradle, PHP memiliki Composer, dan Python memiliki pip. Lalu bagaimana dengan JavaScript? Tentu JavaScript juga punya! NPM adalah salah satu package manager yang banyak digunakan oleh JavaScript developer dalam mengelola package, selain NPM ada juga Yarn Package Manager.

Keduanya, baik yarn atau npm, merupakan package manager yang dapat membantu kita dalam mengembangkan aplikasi web atau node. Namun, pada materi ini kita hanya akan membahas salah satu package managernya saja, yaitu NPM. Hal tersebut karena NPM merupakan package manager standar yang disediakan Node.js.

Node Package Manager (NPM) merupakan pengelola package untuk JavaScript yang dapat memudahkan kita dalam mengelola package yang tersedia pada <https://www.npmjs.com/>. NPM ini merupakan standard package manager yang disediakan oleh Node.js dan otomatis terpasang ketika memasang Node.js pada komputer kita.

Berbicara mengenai package, sebelumnya kita sudah membuat dan mengetahui fungsi module pada JavaScript. Dengan module kita dapat berbagi kode JavaScript untuk digunakan pada berkas yang berbeda. Bahkan, melalui NPM kita dapat menggunakannya pada proyek yang berbeda. Dalam arti lain, package yang tersedia pada NPM adalah sebuah module.

Dalam menggunakan NPM kita dapat menulis perintah pada terminal (Linux/macOS) atau command prompt/PowerShell (Windows). Ada banyak command yang tersedia pada NPM. Untuk melihat daftarnya, kita dapat menjalankan perintah npm help pada terminal/command prompt.

Lalu gunakan -h jika kita ingin melihat panduan penggunaan salah satu perintahnya. Contoh, Anda ingin melihat detail cara penggunaan perintah install, maka untuk melihatnya tuliskan perintah berikut:

1. npm install -h

Hasilnya adalah panduan dalam penggunaan perintah install.

-h tidak hanya dapat digunakan pada perintah install saja, tetapi juga pada seluruh perintah lainnya yang ada pada NPM.

Jika melihat panduan pada gambar di atas, kita bisa menemukan beberapa “aliases”. Aliases atau alias merupakan cara lain dalam melakukan perintah tersebut. Itu berarti perintah install dapat kita tuliskan dengan i, ins, isntal, add, atau yang lainnya.

1. npm install
2. npm i
3. npm in
4. npm ins
5. npm isntal
6. npm add
8. // semua perintah di atas memiliki fungsi yang sama

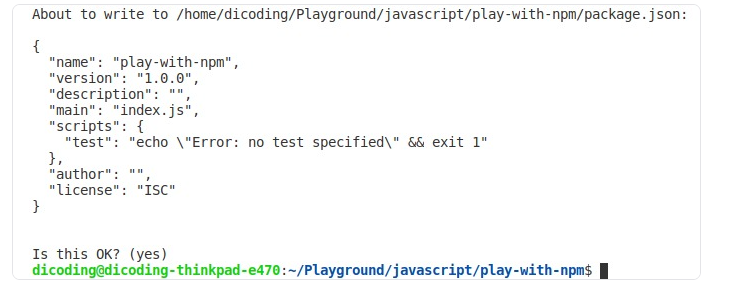
Alias ini dapat berupa cara cepat (shortcut) dan juga kesalahan pengejaan yang umum (install → isntall). Hal tersebut sangat membantu developer agar lebih cepat dalam menuliskan sebuah perintah di command line.

Anda bisa mendapatkan penjelasan lengkap mengenai seluruh perintah yang ada di NPM pada laman resmi berikut: <https://docs.npmjs.com/cli/v7/commands>. Namun, terdapat beberapa perintah penting yang akan biasa kita gunakan dalam materi dan juga pengembangan aplikasi selanjutnya.

| **Command** | **Description** | **Common Options** |
| --- | --- | --- |
| init | Membuat berkas package.json pada project | [--force|-f|--yes|-y|--scope] |
| install <package-name> | Memasang dan mendaftarkan package pada berkas package.json | [-P|--save-prod|-D|--save-dev|-O|--save-optional] [-E|--save-exact] [-B|--save-bundle] [--no-save] [--dry-run] |
| run-script <command> | Menjalankan perintah yang terdapat pada objek scripts di berkas package.json | [--silent] [-- <args>...] |
| uninstall <package-name> | Menghapus dan mengeluarkan package dari berkas package.json | [-S|--save|-D|--save-dev|-O|--save-optional|--no-save] |
| version | Untuk melihat versi package yang tersedia secara global atau lokal | [<newversion> | major | minor | patch | premajor | preminor | prepatch | prerelease [--preid=<prerelease-id>] | from-git] |

## Installing Packages

Setelah mengetahui gambaran singkat mengenai Node Package Manager (NPM), sekarang kita akan mulai menggunakan NPM untuk menambahkan package eksternal ke dalam project aplikasi kita. Anda dapat membuat project baru atau menggunakan project node yang telah dibuat sebelumnya. Pastikan di dalam folder project terdapat berkas **package.json**, jika belum, jalankan perintah npm init pada project Anda.



Sebelum mulai menginstal package, kita perlu tahu terlebih dulu bahwa ada dua jenis instalasi package, yaitu local install dan global install.

Local package akan dipasang di dalam direktori atau folder yang sama dengan project kita. Package ini akan diletakkan di dalam folder **node\_modules**.

Sementara global package dipasang di satu tempat pada sistem perangkat kita (tergantung pengaturan pada saat instalasi node/npm).

Umumnya, semua package harus diinstal secara lokal. Ini memastikan setiap project atau aplikasi di komputer kita memiliki package dan versi yang sesuai dengan kebutuhan. Untuk menginstal package secara lokal caranya sama seperti yang telah kita bahas sebelumnya, yaitu dengan perintah npm install.

1. npm install <package-name>

Lalu, kapan kita menggunakan global package? Sebuah package harus diinstal secara global hanya saat ia menyediakan perintah yang dapat dieksekusi dari CLI dan digunakan kembali di semua project. Beberapa contoh package yang perlu diinstal secara global, antara lain:

* [npm](https://www.npmjs.com/package/npm)
* [nodemon](https://www.npmjs.com/package/nodemon)
* [mocha](https://www.npmjs.com/package/mocha)

Untuk menginstal package secara global, kita cukup menambahkan parameter -g pada perintah npm install. Di mana -g berarti global.

1. npm install -g <package-name>

Kembali ke project kita, mari tambahkan sebuah package. Sebagai contoh, kita akan menggunakan package [lodash](https://www.npmjs.com/package/lodash). Lodash adalah package yang berisi banyak utilitas untuk memudahkan kita dalam menulis kode JavaScript, khususnya yang berhubungan dengan array, object, string, dll.

Jalankan perintah berikut pada terminal/command prompt Anda (pastikan perintah dijalankan di direktori project):

1. npm install lodash

Setelah instalasi berhasil, buka dan periksa file **package.json** Anda. Akan muncul object baru dependencies berisi package yang telah kita tambahkan ke dalam project diikuti dengan versinya.

1. "dependencies": {
2. "lodash": "^4.17.21"
3. }

Sebenarnya terdapat dua tipe object dependencies dalam berkas **package.json**. Yang pertama object dependencies dan yang kedua object devDependencies. Apa perbedaannya?

Objek dependencies merupakan objek yang menampung package yang kita gunakan untuk membuat aplikasi. Sedangkan objek devDependencies digunakan untuk package yang berkaitan hanya saat proses pengembangan aplikasi, contohnya seperti package [jest](https://www.npmjs.com/package/jest) yang digunakan untuk testing. Package seperti jest ini hanya digunakan saat proses pengembangan aplikasi. Ia tidak digunakan lagi ketika aplikasi rilis atau digunakan langsung oleh pengguna.

Untuk memasang package sebagai devDependencies, kita bisa menambahkan parameter --save-dev pada perintah npm install.

1. npm install <package-name> --save-dev

Setelah package terpasang, maka akan ditambahkan ke berkas **package.json**.

1. "devDependencies": {
2. "jest": "^26.6.3"
3. }

Di dalam project akan muncul berkas **package-lock.json**. Berkas ini secara otomatis dibuat oleh Node untuk menjelaskan susunan project dan package (suatu package bisa saja menggunakan package lainnya). Berkas **package-lock.json** juga mendefinisikan versi package yang digunakan dengan lebih spesifik. Apa maksudnya?

Jika diperhatikan, pada file package.json kita menentukan versi dengan simbol caret (^). Selain caret, npm juga menggunakan simbol tilde (~). Keduanya digunakan untuk menentukan versi minor dan versi patch yang akan digunakan.

Jadi, jika kita melihat versi ~1.0.2 itu berarti npm dapat menginstal versi 1.0.2 atau versi patch terbaru seperti 1.0.4. Jika versi package dituliskan dengan caret seperti ^1.0.2, itu berarti npm dapat menginstall versi 1.0.2 atau versi minor terbaru seperti 1.1.0 [[6](https://michaelsoolee.com/npm-package-tilde-caret/)].

File **package-lock.json** akan mendefinisikan secara spesifik mana versi yang akan digunakan.

1. "dependencies": {
2. "lodash": {
3. **"version": "4.17.21",**
4. "resolved": "https://registry.npmjs.org/lodash/-/lodash-4.17.21.tgz",
5. "integrity": "sha512-v2kDEe57lecTulaDIuNTPy3Ry4gLGJ6Z1O3vE1krgXZNrsQ+LFTGHVxVjcXPs17LhbZVGedAJv8XZ1tvj5FvSg=="
6. }
7. }

## Using Package

Ok. Kita sudah belajar bagaimana memasang package pada project sebagai dependencies. Lalu bagaimana cara menggunakan package tersebut?

Perlu kita pahami kembali bahwa package yang kita tambahkan ke dalam project sebenarnya adalah module. Itulah kenapa di dalam project kita akan muncul juga folder **node\_modules**. Di dalamnya berisi kode-kode JavaScript yang menyusun suatu package. Jika Anda “berani”, Anda dapat melihat seperti apa kode di dalam package lodash.

Karena merupakan module, kita bisa menambahkan kode dari package menggunakan keyword **import** seperti yang telah dipelajari pada materi Module.

1. import variableName from 'package-name';

Sehingga, untuk menggunakan kode dari package lodash yang telah kita pasang, tambahkan kode berikut pada berkas **index.js**:

1. import \_ from 'lodash';

Penamaan menggunakan underscore (\_) seperti di atas merupakan standar dari lodash yang perlu kita ikuti.

Sekarang mari kita lihat bagaimana lodash ini bekerja. Di dalam dokumentasinya, lodash menyebutkan bahwa mereka menyediakan utilitas untuk membuat JavaScript lebih mudah dengan menghilangkan kerumitan ketika menggunakan array, number, object, string, dll.

Misalnya, untuk menjumlahkan setiap nilai number di dalam array, lakukan dengan cara seperti berikut.

1. import \_ from 'lodash';
3. const myArray = [1, 2, 3, 4];
4. let sum = 0;
6. for(let i = 0; i < myArray.length; i++) {
7. sum += myArray[i];
8. }
10. console.log(sum);
12. /\* output
13. 10
14. \*/

Atau, kita bisa menggunakan fungsi reduce seperti berikut:

1. import \_ from 'lodash';
3. const myArray = [1, 2, 3, 4];
4. let sum = myArray.reduce((prev, curr) => {
5. return prev + curr;
6. });
8. console.log(sum);
10. /\* output
11. 10
12. \*/

Dengan lodash, kita dapat meringkas kode menjadi seperti ini:

1. import \_ from 'lodash';
3. const myArray = [1, 2, 3, 4];
4. const sum = \_.sum(myArray);
6. console.log(sum);
8. /\* output
9. 10
10. \*/

Anda bisa mengeksplorasi lebih jauh lagi tentang fitur-fitur dari lodash pada halaman [dokumentasinya](https://lodash.com/).

## Uninstall Package

Kita sudah mengetahui cara memasang dan menggunakan package npm. Lalu, bagaimana caranya untuk menghapus package yang sudah tidak gunakan?

Untuk melakukannya cukup mudah. Jika package berada pada objek dependencies, kita dapat menghapusnya menggunakan perintah:

1. npm uninstall <package-name>

Jika package terdapat pada devDependencies, kita cukup menambahkan --save-dev di akhir perintahnya.

1. npm uninstall <package-name> --save-dev

Contoh, jika kita ingin menghapus package lodash dari project, berarti tuliskan perintah:

1. npm uninstall lodash

Dengan begitu, package lodash akan dihapus dari objek dependencies pada berkas **package.json**.

1. {
2. "name": "play-with-npm",
3. "version": "1.0.0",
4. "description": "",
5. "main": "index.js",
6. "scripts": {
7. "test": "echo \"Error: no test specified\" && exit 1"
8. },
9. "author": "",
10. "license": "ISC"
11. }

Selain menghapus nilainya dari **package.json**, perintah tersebut juga akan menghapus semua berkas yang berhubungan dengan package lodash pada folder **node\_modules**.

Setelah menghapus package, tentu fungsi-fungsi dari package lodash sudah tidak dapat kita gunakan lagi.

Perintah uninstall juga memiliki beberapa alias, sehingga dapat lebih cepat dalam menuliskan perintahnya. Anda bisa melihat apa saja alias yang dapat digunakan dengan menjalankan perintah:

1. npm uninstall -h

## Rangkuman Materi

Pada modul ini kita telah membahas seputar Node Package Manager (NPM) untuk mengelola kode atau package milik orang lain ke dalam project kita. Seharusnya Anda sudah bisa menggunakan perintah dasar NPM seperti:

* Menginstal package melalui NPM.
* Menggunakan package untuk membantu pengembangan aplikasi.
* Menghapus proyek yang sudah tidak digunakan.

Aturan

Knowledge Check : Node Package Manager bertujuan untuk menguji pengetahuan Anda tentang materi yang telah dipelajari di modul ini.

Terdapat 2 pertanyaan yang harus dikerjakan dalam kuis ini. Beberapa ketentuannya sebagai berikut:

* Syarat nilai kelulusan : 100%
* Durasi ujian : 5 menit

Apabila tidak memenuhi syarat kelulusan, Anda bisa mengulang pengerjaan kuis kembali.

Selamat Mengerjakan!

# [JavaScript Testing](https://www.dicoding.com/academies/256/tutorials/14077)

## Pengenalan JavaScript Testing

Setelah kita mempelajari tentang dasar-dasar dalam pemrograman JavaScript, seperti menuliskan perintah dan menerapkan paradigma-paradigma yang populer maka proses selanjutnya adalah publikasi ke pengguna. Namun, sebelum sebuah program dipublikasikan, seharusnya sebuah program harus melalui tahap pengujian terlebih dahulu. Proses pengujian tersebut digunakan untuk melakukan kontrol kualitas dari kode yang kita tulis, memastikan bahwa software yang akan dipublikasikan ke pengguna minim adanya kecacatan atau bugs. Oleh sebab itu, sebagai seorang pemrogram kita harus melakukan testing saat proses pengembangan perangkat lunak.

Sebuah proses pengujian perangkat lunak dapat dilakukan secara:

* **Manual**  
  Proses pengujian secara manual oleh seorang yang ditunjuk sebagai test, atau sebagian pengguna yang memang mendapatkan akses untuk pengujian pre-release. Proses ini biasanya berkaitan dengan usability, accessibility dari sebuah aplikasi.
* **Otomatis**  
  Proses pengujian secara otomatis dilakukan oleh komputer dengan menuliskan script khusus, biasanya dilakukan oleh software engineer langsung ataupun oleh seorang QA Engineer. Proses ini biasanya berkaitan dengan fungsionalitas dari sebuah aplikasi.

Dari proses pengujian di atas, jenis-jenis pengujian dalam pengembangan perangkat lunak secara umum dapat dibagi menjadi 4 jenis pengujian, yaitu:

* **Static test**  
  Memastikan tidak adanya typo (naming convention yang standar) dan memastikan tidak ada error types.
* **Unit test**  
  Dilakukan untuk memastikan bahwa setiap unit kode yang kita tulis sudah bisa bekerja sesuai harapan. Unit sendiri berarti komponen terkecil yang dapat diuji secara terisolasi dalam perangkat lunak yang kita buat, dapat berupa fungsi bahkan kelas jika kita menggunakan paradigma OOP. Proses ini juga dapat diautomasikan.
* **Integration test**  
  Memastikan beberapa serangkaian fungsi yang saling ketergantungan satu sama lain berjalan semestinya. Proses pengujian ini dapat dilakukan secara otomatis dengan menuliskan script test.
* **End-to-End test**  
  Proses pengujian sebuah aplikasi untuk menguji flow dari awal hingga akhir, layaknya seorang user saat menggunakan aplikasi. Memastikan bahwasanya aplikasi berfungsi selayaknya. Biasanya proses ini dapat dilakukan secara otomatis maupun manual oleh tester.

Dalam modul ini, kita akan mempelajari bagaimana cara membuat sebuah pengujian secara otomatis. Proses automasi ini digunakan untuk memudahkan proses pengujian yang repetitive terhadap sebuah kasus umum. Sehingga kita tidak perlu lagi melakukan uji secara manual satu persatu pada sebuah fungsi atau perintah.

Ketika kita menulis sebuah kasus pengujian (test case), maka ada beberapa poin yang harus kita definisikan terlebih dahulu:

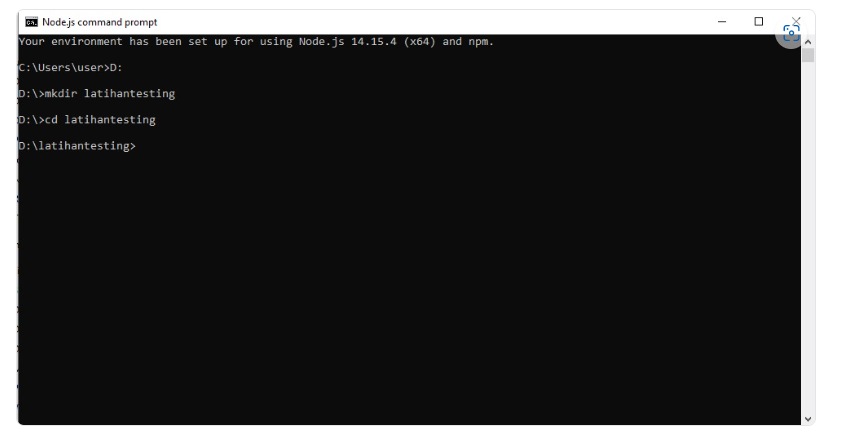
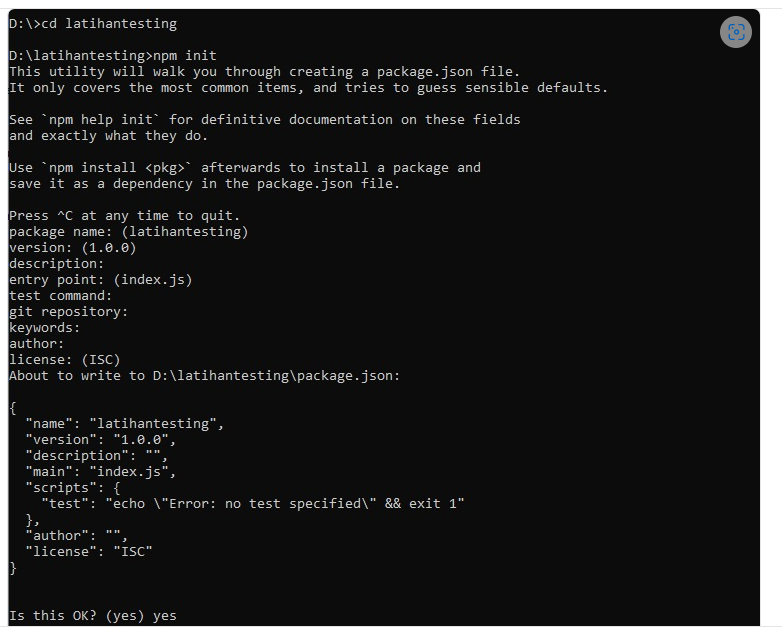
* Apa yang ingin diuji?  
  Misalkan: Melakukan uji pada fungsi menghitung rata-rata nilai siswa, atau dapat juga melakukan pengujian pada proses registrasi akun, dan lain sebagainya.
* Ekspektasi apa yang diharapkan?
  + Untuk kasus menghitung nilai rata-rata:
    - Menghasilkan output perhitungan yang sesuai berdasarkan input yang diberikan.
    - Input harus berupa angka.
  + Untuk kasus proses registrasi akun:
    - Pengguna dapat melakukan registrasi secara normal.
    - Pengguna dengan email sama tidak dapat melakukan registrasi.
    - dan lain sebagainya.

## Jest

Dalam penulisan kode pengujian, kita dapat menggunakan library tambahan untuk mempermudah penulisan kode pengujian.

Jest merupakan salah satu framework testing paling populer untuk menuliskan kode pengujian pada bahasa pemrograman JavaScript. Jest dapat digunakan untuk menuliskan script testing pada aplikasi backend maupun frontend.

Kita akan mencoba untuk menuliskan sebuah kode pengujian menggunakan framework jest.

1. Membuat proyek baru dengan direktori bernama **latihantesting**.  
    
2. Setelah masuk ke direktori tersebut, kita dapat melakukan init project kita.  
    
3. Setelah proyek tersebut diinisialisasi. Selanjutnya kita dapat melakukan instalasi library framework jest.
   1. npm install --save-dev jest
4. Setelah proses instalasi selesai, buka proyek di code editor.
5. Pada file **package.json**, kita tambahkan script untuk test dengan cara menambahkan baris kode berikut:
   1. {
   2. "scripts": {
   3. "test": "jest"
   4. }
   5. }

Penambahan script tersebut digunakan agar kita bisa menjalankan script test yang nanti kita buat menggunakan runner.

## Menulis Kode Pengujian

Dari proyek sebelumnya, kita akan coba untuk memulai pengenalan format sederhana saat kita akan menuliskan sebuah test. Perhatikan kode di bawah ini.

1. test('deskripsi dari testcase kamu', () => {
2. expect(perintah).matcher(nilai yang diekspektasikan);
3. });
5. // contoh
6. test('dua tambah dua adalah empat', () => {
7. expect(2+2).toBe(4);
8. });

Dari contoh kode di atas expect(2 + 2) merupakan segmen yang berisi perintah yang menghasilkan nilai balikan, perintah tersebut dapat berupa fungsi ataupun perintah langsung.

toBe(4) disebut dengan matcher, berisi nilai yang diharapkan dari sebuah perintah yang dilakukan sebelumnya. Dalam jest sendiri terdapat berbagai matchers yang dapat digunakan, misalkan:

|  |  |
| --- | --- |
| toBe(x) | bahwa nilai ekspektasi yang diharapkan adalah x |
| toEqual(x) | bahwa nilai ekspektasi yang diharapkan sama dengan x |
| toBeNull() | bahwa nilai ekspektasi yang diharapkan adalah null |
| toBeTruthy() | bahwa nilai ekspektasi yang diharapkan adalah truth |
| toBeFalsy() | bahwa nilai ekspektasi yang diharapkan adalah false |

Selain matchers di atas, Anda juga dapat melihat matcher lain yang dapat digunakan di sini <https://jestjs.io/docs/using-matchers>.

Pada framework jest, kita juga dapat menuliskan sebuah test secara grouping berdasarkan karakteristik ataupun grup yang sama menggunakan segmen describe , misalkan sebagai berikut:

1. describe('pengujian olah aritmatika', () => {
2. test('#1 dua tambah dua adalah empat', () => {
3. expect(2+2).toBe(4);
4. });
6. test('#2 dua kali tiga adalah enam', () => {
7. expect(2\*3).toBe(6);
8. });
9. });

## Pengujian Kode

Kita telah mempelajari bentuk penulisan script test menggunakan Jest. Selanjutnya, kita akan membuat 2 buah fungsi untuk pengujian kode berdasarkan proyek yang sudah dibuat di materi sebelumnya. Perhatikan urutan langkahnya di bawah ini.

Buatlah file bernama **gradeCalculations.js** dalam folder **latihantesting**.

1. const averageExams = (valuesExam) => {
2. const numberValidation = valuesExam.every(exam => typeof exam === 'number');
3. if (!numberValidation) throw Error('please input number');
5. const sumValues = valuesExam.reduce((accumulator,currentValue) => accumulator + currentValue, 0);
6. return sumValues / valuesExam.length;
7. };
9. const isStudentPassExam = (valuesExam, name) => {
10. const minValues = 75;
11. const average = averageExams(valuesExam);
13. if (average > minValues) {
14. console.log(`${name} pass the exams`);
15. return true;
16. } else {
17. console.log(`${name} fail the exams`);
18. return false;
19. }
20. };
22. module.exports = { averageExams, isStudentPassExam };

Kode di atas terdapat 2 fungsi:

* **averageExams** : Menghitung rata-rata nilai siswa dari sebuah inputan berupa daftar angka dalam bentuk array.
* **isStudentPassExam** : Melakukan kalkulasi apakah seorang siswa lulus ujian atau tidak berdasarkan nilai rata-rata yang didapatkan (bergantung pada fungsi averageExams).

Dari kedua contoh kode di atas, kita akan melakukan simulasi pengujian unit test & integration test.

Penulisan kode pengujian dapat dilakukan dengan memulai kerangka testcase yang sudah kita singgung diatas.

* Apa yang akan diuji?
  + Fungsi menghitung rata-rata.
  + Fungsi untuk mengecek apakah seorang siswa lulus ujian.
* Ekspektasi yang diharapkan?
  + Menghasilkan nilai yang sesuai dari inputan.
  + Inputan harus berupa angka.
  + Menghasilkan hasil kelulusan iya/tidak pada siswa dengan nilai kriteria tertentu.

Setelah kita menuliskan contoh kode **gradeCalculations.js**, hal yang perlu dilakukan  pada penulisan kode pengujian menggunakan jest adalah sebagai berikut:

* Simpan script test dalam sebuah folder bernama **\_\_tests\_\_**.
* 1 file test untuk 1 lingkup perintah dengan format **namaScopeFunctions.test.js**.

Dari contoh di atas maka semua testcase dari gradeCalculations ditulis di dalam **gradeCalculations.test.js**.

Penulisan unit test untuk kode averageExams sebagai berikut:

1. const { averageExams } = require('../gradeCalculations');
3. test('it should return exact average', () => {
4. const listValueOfExams = [80, 100, 100, 80];
5. expect(averageExams(listValueOfExams)).toEqual(90);
6. })

Untuk menjalankan kode pengujian tersebut, dapat menjalankan perintah runner yang sudah kita buat sebelumnya.

1. npm run test

Dari contoh di atas, kita dapat melihat bahwa test case tersebut sesuai ekspektasi. Selain itu, kita juga dapat membuat multiple case dengan membuat tes yang lain, misalnya dengan kode berikut:

1. test('it should handle non-number ', () => {
2. const listValueOfExams = [80, 'a', '100', 80];
3. expect(() => averageExams(listValueOfExams)).toThrow();
4. })

Integration test dapat dijalankan ataupun ditulis bersamaan dengan unit test. Sehingga kita dapat melakukan grouping dari contoh kode diatas menjadi sebagai berikut:

1. const { averageExams, isStudentPassExam } = require('../gradeCalculations');
3. describe('grade calculations', () => {
4. test('it should return exact average', () => {
5. const listValueOfExams = [80, 100, 100, 80];
6. expect(averageExams(listValueOfExams)).toEqual(90);
7. });
9. /\*\*
10. \* Integration testing
11. \*/
12. test('it should return exam passed status', () => {
13. const listValueOfExams = [80, 100, 100, 80];
14. expect(isStudentPassExam(listValueOfExams, 'Budi')).toEqual(true);
15. })

18. test('it should return exam failed status', () => {
19. const listValueOfExams = [50, 40, 70, 80];
20. expect(isStudentPassExam(listValueOfExams, 'Budi')).toEqual(false);
21. })
22. })

Setelah menuliskan semua test case yang ada, dengan jest kita dapat melihat laporan kode yang sudah terdapat pengujian dengan cara sebagai berikut:

1. npm run test -- --coverage

Setelah prosesnya selesai, jest akan secara otomatis men-generate laporan pengujian dalam folder **coverage**. Dalam folder tersebut terdapat:

* **index.html** berisi laporan test secara keseluruhan (1 proyek).
* **fungsiDiTest.js.html** berisi laporan test per kode pengujian.

## Rangkuman Materi

Proses pengujian yang dilakukan secara automate memiliki kelebihan dan kekurangan masing-masing. Pembuatan kode pengujian akan menyita lebih banyak waktu saat proses pengembangan (di awal) perangkat lunak, dibandingkan jika kita tidak melakukan pengujian. Namun, proses pengujian yang dilakukan akan memastikan bahwasanya perangkat lunak yang kita bangun minim akan adanya kecacatan ketika perangkat lunak sudah digunakan oleh pengguna.

Kecacatan/bugs yang ditemukan setelah software dikonsumsi oleh pengguna, proses penelusuran kecacatan, sampai dengan perbaikan akan lebih memakan waktu. Selain itu, kita tidak dapat memastikan, bahwasanya perbaikan yang kita lakukan tidak menambah bugs baru ditempat yang lain.

Oleh karena itu, penulisan script kode pengujian sangat membantu, selain memastikan kualitas kode yang dibuat minim adanya kecacatan. Juga berimplikasi pada biaya waktu (time cost) pada pengembangan perangkat lunak.

### Referensi

* [Cheatsheet Jest](https://devhints.io/jest)

Aturan

Knowledge Check : JavaScript Testing bertujuan untuk menguji pengetahuan Anda tentang materi yang telah dipelajari di modul ini.

Terdapat 2 pertanyaan yang harus dikerjakan dalam kuis ini. Beberapa ketentuannya sebagai berikut:

* Syarat nilai kelulusan : 100%
* Durasi ujian : 5 menit

Apabila tidak memenuhi syarat kelulusan, Anda bisa mengulang pengerjaan kuis kembali.

Selamat Mengerjakan!

# [Penutup](https://www.dicoding.com/academies/256/tutorials/14092)

## [[Ujian Akhir](https://www.dicoding.com/academies/256/tutorials/14092)](https://www.dicoding.com/academies/256/tutorials/14097)

asa