**Dasar-Dasar Node.js untuk Back-End**

Setelah Anda mengetahui teori tentang back-end, server, pola komunikasi server dengan client, serta REST API, *what’s next*? Yup! selanjutnya untuk menjadi seorang Back-End Developer tentu Anda perlu mengetahui bekal yang harus dimiliki dalam membangun sebuah aplikasi back-end atau web service.

Kita akan menggunakan Node.js dalam membangun web service. Namun sebelum menyelami materi ini lebih jauh, mari kita pelajari terlebih dahulu beberapa dasar-dasar API yang ada pada Node.js. Beberapa dasar yang akan dipelajari kali ini akan sangat membantu kita dalam pengembangan web service nantinya.

Pada akhir modul ini, diharapkan Anda dapat:

* Mengetahui apa itu Node.js
* Mengeksekusi JavaScript dengan Node.js
* Mengetahui Node.js Global Object
* Memahami Modularization
* Menggunakan Node Package Manager
* Memahami Eventing
* Memahami Filesystem
* Memahami teknik Readable Stream dan Writable Stream

Tools Belajar Dasar-Dasar Node.js untuk Back-End

Modul kali ini memiliki prasyarat sebelum Anda mengikutinya. Selain kemampuan JavaScript, terdapat dua tools yang perlu Anda siapkan, yakni Text Editor dan Node.js. Bila Anda sudah mengikuti seluruh latihan pada kelas Belajar Dasar Pemrograman JavaScript, tools ini seharusnya sudah terpasang pada komputer Anda. Bila belum, silakan unduh dan pasang dulu yah.

Text Editor

Selama mengikuti kelas ini, kami merekomendasikan Anda untuk menggunakan VSCode. Inilah text editor yang sangat populer dan gratis untuk digunakan. Selain itu, text editor ini memiliki plugin berlimpah yang dapat membuat fungsionalitas menjadi lebih kaya lagi. Visual Studio Code dapat dijalankan pada sistem operasi Windows, macOS, ataupun Linux. Untuk mengunduhnya, silakan kunjungi laman unduh visual studio code.

Node.js

Pastikan komputer Anda sudah terpasang Node.js dengan minimal versi 18 atau lebih tinggi. Untuk mengetahui versi Node.js yang terpasang, silakan tulis perintah ini pada Terminal atau CMD.

node -v

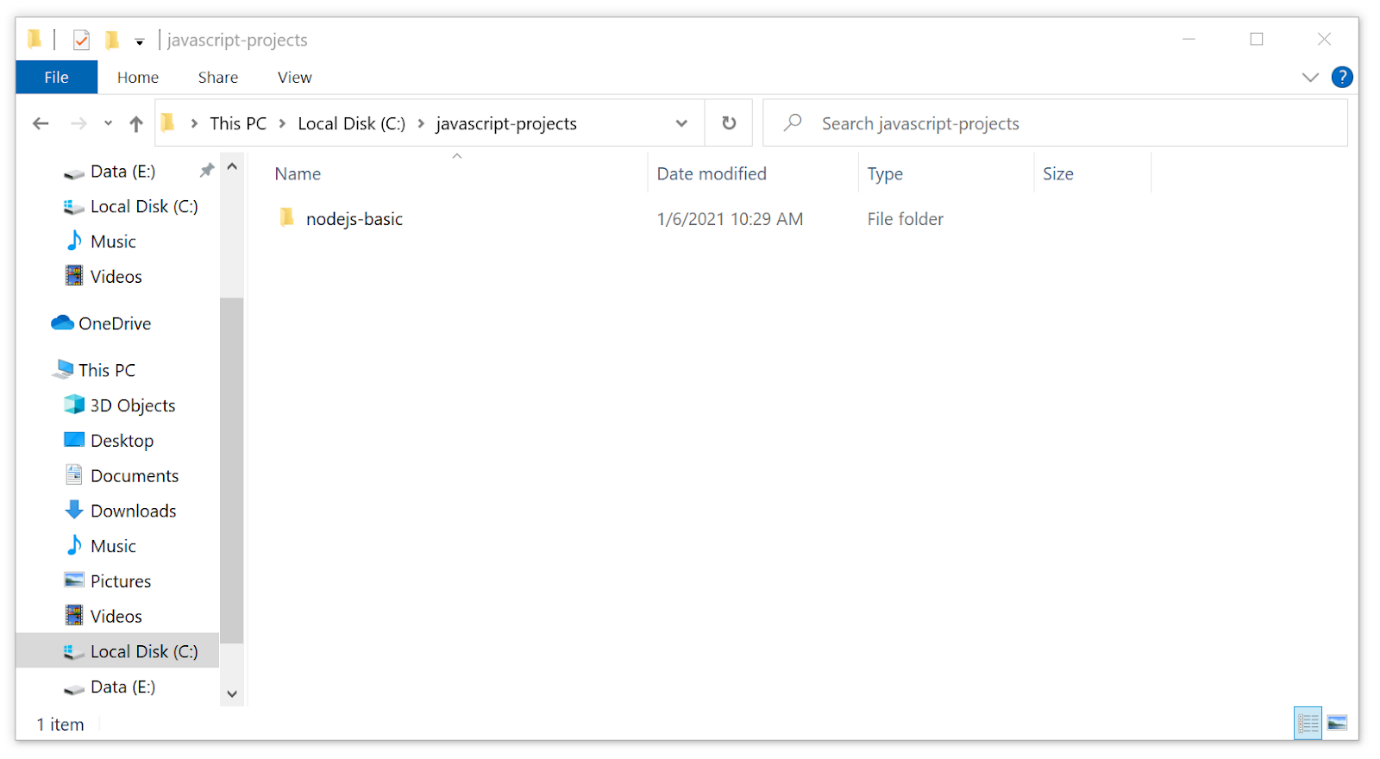
Sudah siap dengan prasyarat yang ada? Jika ya, yuk lanjut ke materi selanjutnya yah!

**Latihan: Membuat Proyek Node.js**

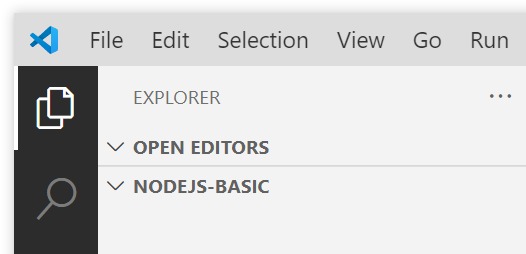
Setelah mengenal Node.js kini saatnya kita belajar cara menggunakannya. Kita akan mulai dari membuat proyek Node.js, menjalankan kode JavaScript menggunakan Node.js, hingga mempelajari berbagai API yang ada di dalamnya. Jadi sekali lagi pastikan prasyarat yang ada sudah Anda penuhi yah. Siapkan Text Editor, dan kita akan mulai menuliskan kode JavaScript.

Ikuti langkah-langkah berikut untuk membuat proyek Node.js.

1. Sebelum membuat proyek, buatlah folder baru terlebih dahulu. Folder ini akan digunakan sebagai tempat penyimpanan berkas proyek dan JavaScript yang kita tulis nanti. Kami sarankan, Anda buat folder tersebut di alamat C -> javascript-projects -> nodejs-basic bagi pengguna Windows; home -> javascript-projects -> nodejs-basic bagi pengguna Linux atau macOS.  
   * [**Windows**](https://www.dicoding.com/academies/261/tutorials/14202?from=14197#tab1-code1)
   * [Linux](https://www.dicoding.com/academies/261/tutorials/14202?from=14197#tab1-code2)
   * [macOS](https://www.dicoding.com/academies/261/tutorials/14202?from=14197#tab1-code3)

[](https://www.dicoding.com/academies/261/tutorials/14202?from=14197)

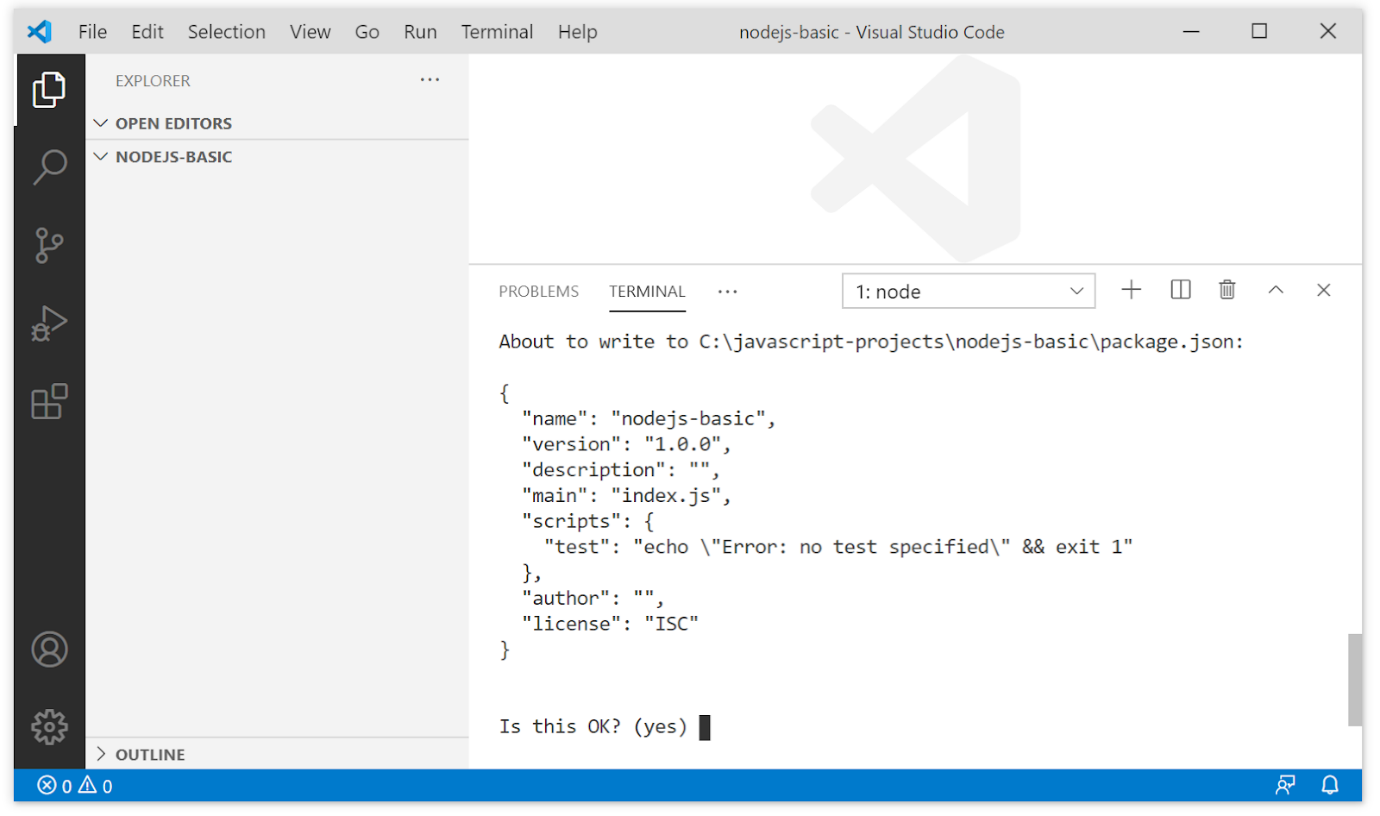
1. Selanjutnya, buka folder nodejs-basic menggunakan VSCode. Caranya, pada Visual Studio Code pilih menu File -> Open Folder -> [pilih foldernya]. Folder pun berhasil terbuka melalui VSCode.

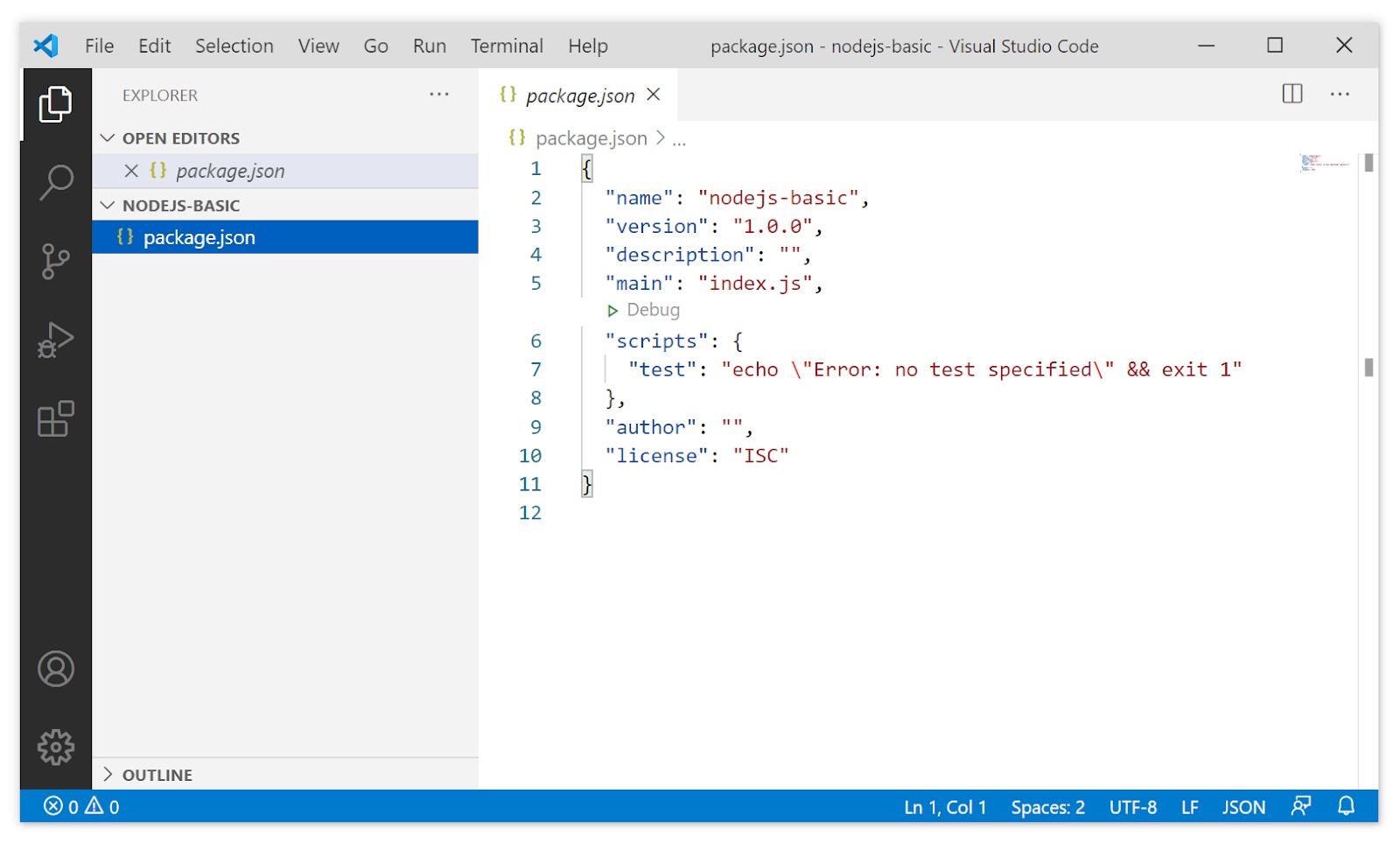
[](https://www.dicoding.com/academies/261/tutorials/14202?from=14197)

1. Untuk membuat proyek JavaScript, silakan buka Terminal pada VSCode. Pilih menu Terminal -> New Terminal, kemudian tuliskan perintah:
2. npm init

NPM alias Node Package Manager merupakan JavaScript Package Manager bawaan dari Node.js. Melalui NPM ini kita dapat membuat Node.js package (proyek) dan mengelola penggunaan package eksternal yang digunakan. Kita akan membahas NPM lebih detail nanti.

Jika Anda yang tidak menggunakan Visual Studio Code, gunakan Terminal/Command Prompt usungan OS Anda. Namun, sesuaikan lokasinya pada folder proyek ya.

1. Setelah menuliskan perintah di atas, Anda akan diberikan beberapa pertanyaan untuk mengisi nilai package name, version, description. Semua itu merupakan informasi dasar dari aplikasi yang Anda buat.  
   Nilai yang berada di dalam tanda kurung merupakan nilai default. Anda dapat menggunakan nilainya dengan langsung menekan tombol Enter. Untuk saat ini, cukup berikan semua pertanyaan dengan nilai default.
2. Setelah mengisi seluruh pertanyaan yang diberikan, Anda akan diberitahu untuk melihat hasil akhir yang dibuat pada berkas package.json.  
   [](https://www.dicoding.com/academies/261/tutorials/14202?from=14197)

Jika nilai yang ditampilkan sudah sesuai, langsung saja tekan tombol Enter. Berkas package.json pun telah berhasil dibuat pada proyek kita.        [](https://www.dicoding.com/academies/261/tutorials/14202?from=14197)

Voila! Anda berhasil membuat proyek Node.js.

**Apa itu Node.js**

Dari dulu hingga kini, browser menjadi tempat satu-satunya yang dapat mengeksekusi kode JavaScript. Karenanya, Web Developer perlu mempelajari bahasa pemrograman yang berbeda untuk mengembangkan aplikasi Front-End dan Back-End. Meskipun secanggih dan sekuat apa pun JavaScript berkembang, ia hanya akan digunakan di sisi Front-End saja.

Sebenarnya banyak developer yang mencoba membuat teknologi agar JavaScript dapat dijalankan di luar browser. Namun belum ada yang berhasil. Hingga pada tahun 2009, Ryan Dahl berhasil menciptakan [Node.js](https://nodejs.org/), teknologi yang diharapkan oleh banyak web developer. Tak disangka saat ini teknologi yang diciptakannya menuai popularitas tinggi. Node.js banyak digunakan oleh perusahaan besar sekelas Netflix, Uber, Paypal, dan eBay.

**Node.js berhasil menjadi JavaScript Runtime yang dapat mengeksekusi kode JavaScript di luar browser.** Node.js seolah-olah menjadi gerbang bagi para JavaScript Developer untuk mengembangkan sistem di luar dari browser. JavaScript menjadi bahasa multiplatform yang banyak menggiring developer untuk menggunakannya. Popularitas JavaScript pun meroket! Pada tahun 2014 hingga 2020 JavaScript menjadi bahasa pemrograman nomor satu yang banyak digunakan oleh developer. [[1]](https://octoverse.github.com/)

JavaScript menjadi salah satu pilihan tepat dalam membangun web server, terlebih bila Anda adalah seorang Front-End Web Developer. Anda tentu tidak perlu menggunakan bahasa yang berbeda dalam membangun Back-End. Anda bisa menjadi Full-Stack Developer dengan mempelajari satu bahasa pemrograman saja.

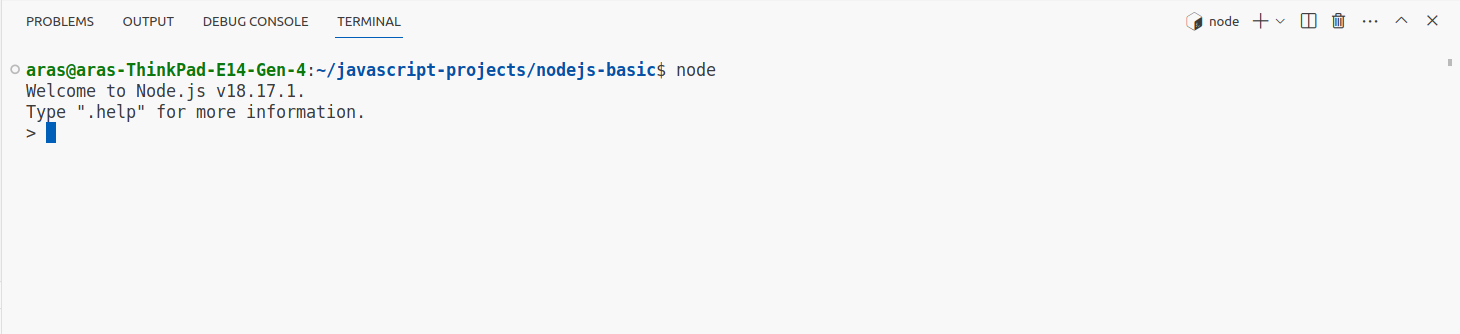
**Latihan: Menjalankan JavaScript Menggunakan Node.js**

Terdapat dua cara dalam menjalankan kode JavaScript menggunakan Node.js. Yang pertama dengan memanfaatkan Node REPL dan yang kedua dengan mengeksekusi berkas berekstensi JS. Mari kita kupas keduanya!

**The Node.js REPL**

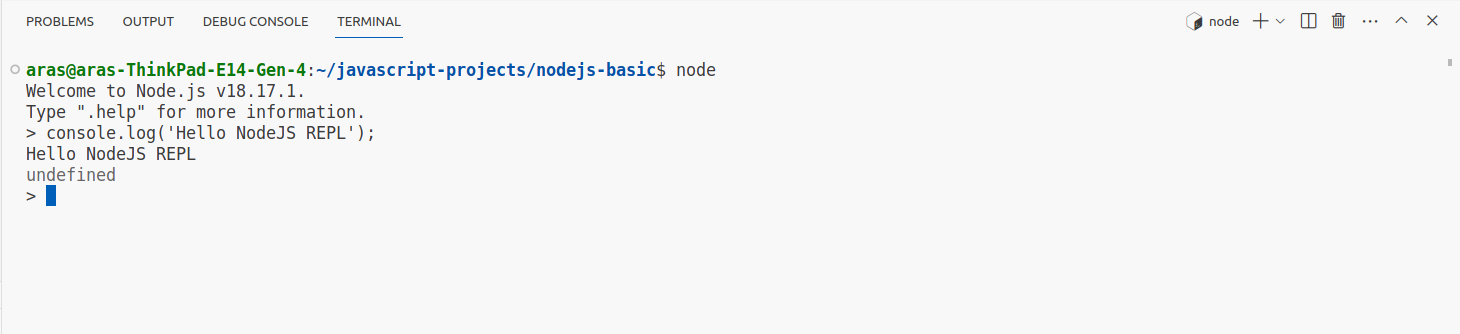
Node.js memiliki fitur REPL atau **R**ead-**E**val-**P**rint **L**oop. Sesuai namanya, fitur ini berfungsi untuk membaca kode JavaScript, mengevaluasi kode tersebut, kemudian mencetak hasil evaluasinya ke console. Nah, untuk *loop,* berarti proses tersebut selalu berulang.

1. REPL merupakan fitur bawaan dari Node.js. Anda bisa mengaksesnya menggunakan perintah **node** pada Terminal.

[](https://www.dicoding.com/academies/261/tutorials/14207?from=14202)

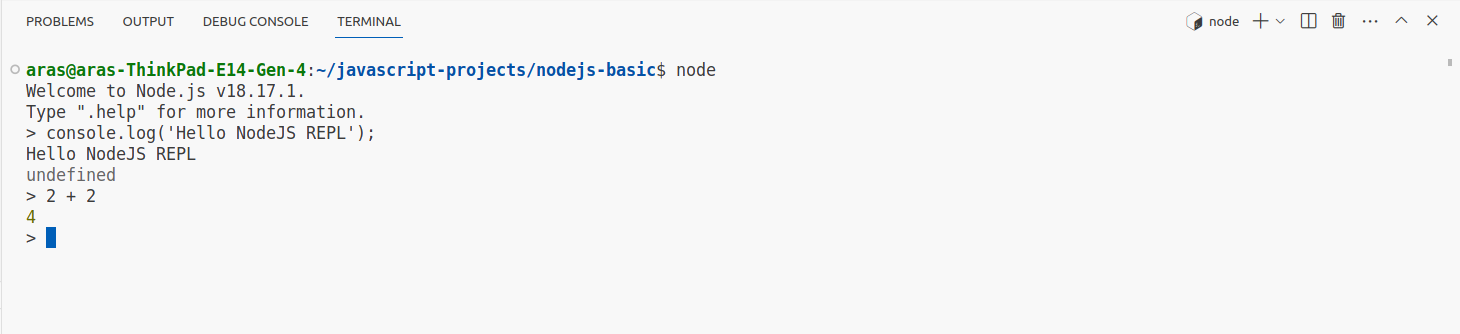
Tanda **>** pada Terminal menunjukan Anda sudah masuk ke mode Node REPL.

1. Sekarang, Anda bisa menuliskan kode JavaScript dan mengeksekusinya dengan menggunakan *enter*.

[](https://www.dicoding.com/academies/261/tutorials/14207?from=14202)

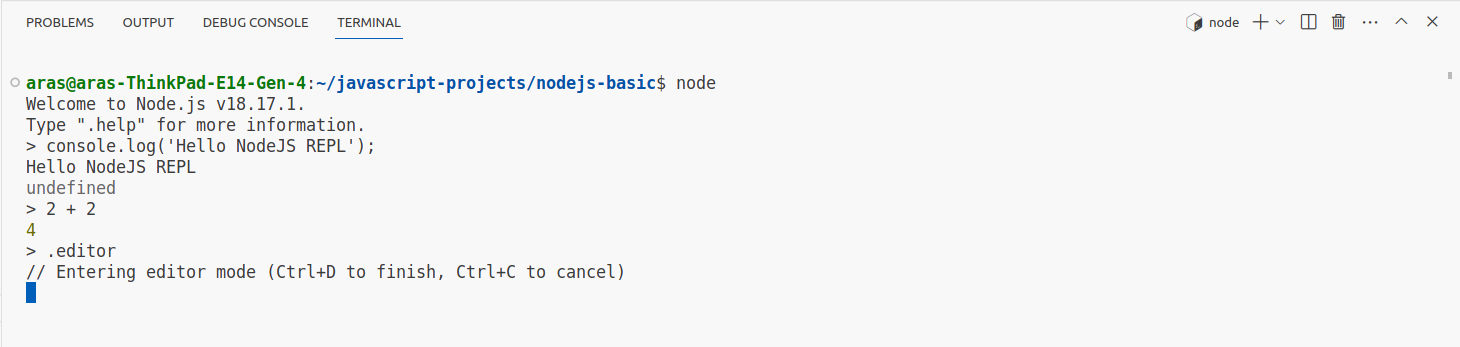
Lihat gambar di atas. Ketika mengeksekusi **console.log(‘Hello Node.js REPL’),**selain pesan “Hello Node.js REPL”, nilai **undefined** juga tercetak. Hal tersebut karena REPL selalu menampilkan nilai evaluasi pada console. Karena method **console.log()** tidak mengembalikan nilai, jadi *undefined*-lah yang tercetak pada console.

1. Untuk membuktikan hal itu, cobalah Anda tuliskan statement yang mengembalikan nilai. Contoh sederhananya **2+2**. Maka nilai **4** akan tercetak pada console.

[](https://www.dicoding.com/academies/261/tutorials/14207?from=14202)

Cukup asik kan fitur REPL? Tapi kok terkesan hanya dapat mengeksekusi kode satu baris saja ya? Bila Anda beranggapan seperti itu, sebenarnya tidak tepat karena di dalam REPL terdapat mode editor yang berfungsi untuk menuliskan kode JavaScript lebih dari satu baris.

1. Untuk menggunakan mode editor, Anda bisa tuliskan perintah **.editor**.

[](https://www.dicoding.com/academies/261/tutorials/14207?from=14202)

Ketika masuk ke mode editor, Anda bisa secara leluasa menuliskan kode JavaScript lebih dari satu baris menggunakan **enter**. Fungsi untuk mengeksekusi kode digantikan dengan kombinasi tombol **CTRL+D.**Untuk keluar dari mode editor, gunakan kombinasi **CTRL+C**.

[](https://www.dicoding.com/academies/261/tutorials/14207?from=14202)

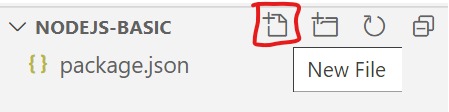
Nilai variabel yang Anda buat di REPL dapat diakses selama Anda masih berada di dalam REPL. Jika Anda menutup Terminal atau keluar dari REPL menggunakan perintah **.exit**, variabel yang sudah Anda buat sebelumnya tidak bisa diakses kembali. Itu artinya, REPL hanya menyimpan memory ketika session masih berlangsung.

Fitur REPL sangat berguna ketika Anda hendak melakukan kalkulasi sederhana, bereksperimen, atau belajar potongan kode JavaScript. Karena melalui REPL Anda bisa mengeksekusi kode JavaScript dan mendapatkan hasil dengan cepat tanpa harus membuat berkas JavaScript terlebih dahulu.

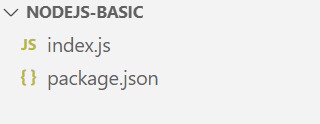
**Running JavaScript File using Node.js**

Cara lain untuk mengeksekusi kode JavaScript menggunakan Node.js adalah melalui berkas JS.

1. Silakan buat berkas JavaScript pada proyek nodejs-basic. Gunakan VSCode agar lebih mudah yah.

[](https://www.dicoding.com/academies/261/tutorials/14207?from=14202)

1. Buatlah berkas JavaScript dengan nama **“index.js”**.

[](https://www.dicoding.com/academies/261/tutorials/14207?from=14202)

1. Di dalam berkas **index.js**, Anda bisa menuliskan kode JavaScript sesuka Anda. Pastikan kode yang Anda tulis menampilkan nilai di console yah, jadi Anda bisa melihat nilai yang tampak pada console. Jika bingung, silakan tuliskan saja kode berikut.
2. **const** message = (name) => {
3. console.log(`Hello ${name}`);
4. }
6. message('JavaScript');
7. Untuk mengeksekusi kode tersebut, silakan buka kembali Terminal. Kemudian, tuliskan perintah:
8. node index.js

Node.js pun akan mengeksekusi berkas**‘index.js’**. Bila Anda menuliskan kode seperti yang dicontohkan di atas, maka akan muncul teks ‘Hello JavaScript’ pada console.

**Node.js Global Object**

JavaScript hanyalah bahasa pemrograman. Ia tidak mengetahui apakah Anda menjalankannya menggunakan browser atau Node.js. Di browser, JavaScript dapat mengontrol fungsionalitas browser seperti mengunjungi halaman, memuat ulang, menutup tabs, serta menampilkan *alert dialog*. JavaScript mampu melakukan itu karena browser menambahkan objek window pada JavaScript.

Di Node.js pun demikian, ia menambahkan objek global guna memberikan fungsionalitas lebih pada JavaScript. Hal ini bertujuan untuk mendukung pengembangan pada environment-nya. Contoh, melalui objek global kita dapat melihat berapa CPU yang digunakan pada komputer, modularisasi berkas JavaScript, menampilkan nilai pada console, dan hal lainnya.

Objek window pada browser dan objek global pada Node.js merupakan *Global Object*. Seluruh fungsi atau properti yang menjadi member dari global object dapat diakses di mana saja alias memiliki cakupan global. Pada Node.js Anda bisa melihat apa saja yang termasuk member dari global objek dengan menggunakan kode berikut:

1. Object.getOwnPropertyNames(global);

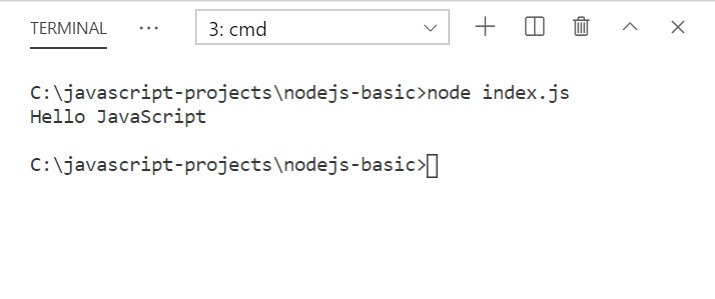
Coba jalankan pada REPL. Ia akan mengembalikan semua *member*-nya.

Banyak sekali yah member dari global objek. Namun dilansir dari website Node.js, sebenarnya mereka hanya menambahkan beberapa objek saja. Objek tersebut dinamakan dengan ‘true globals’. [[2]](https://nodejs.org/api/globals.html#:~:text=Class%3A%20WritableStreamDefaultWriter-,global%20objects)  
Berikut adalah daftarnya:

* global : Global namespace. Member apa pun di dalam object ini dapat diakses pada cakupan global.
* process : menyediakan interaksi dengan proses Node.js yang berjalan.
* console : menyediakan berbagai fungsionalitas [STDIO](http://www.cplusplus.com/reference/cstdio/).
* setTimeout, clearTimeout, setInterval, clearInterval.

Ada juga objek yang merupakan ‘pseudo-globals’ atau objek global semu. Objek ini tidak terlihat bila dicetak menggunakan Object.getOwnPropertyNames(global) sebab ia bukan member langsung dari objek global, melainkan diturunkan dari cakupan module. Karena pada Node.js semua berkas JavaScript adalah module [[3]](https://javascript.info/modules-intro), jadi pseudo-globals dapat diakses layaknya global objek.  
Berikut adalah daftarnya:

* module : digunakan untuk sistem modularisasi pada Node.js.
* \_\_filename : keyword untuk mendapatkan lokasi berkas JavaScript yang dieksekusi. Keyword ini tidak tersedia pada Node.js REPL.
* \_\_dirname : keyword untuk mendapatkan root directory dari berkas JavaScript yang dieksekusi.
* require : digunakan untuk mengimpor module JavaScript.

[](https://www.dicoding.com/academies/261/tutorials/14207?from=14202)

**Modularization**

Semakin kompleks program yang dikembangkan, semakin kompleks juga kode yang dituliskan. Jika kode dituliskan dalam satu berkas saja, maka akan sangat sulit untuk membaca serta memelihara kode tersebut. Idealnya, satu berkas JavaScript hanya memiliki satu tanggung jawab saja. Bila lebih dari satu, itu berarti Anda perlu berkenalan dengan modularisasi.

Modularisasi dalam pemrograman merupakan teknik pemisahan kode menjadi modul-modul yang bersifat independen namun bisa saling digunakan untuk membentuk suatu program yang kompleks. Pemisahan kode menjadi modul-modul terpisah inilah yang dapat membuat kode JavaScript lebih mudah diorganisir.

Pada Node.js, setiap berkas JavaScript adalah modul. Anda bisa membagikan nilai variabel, objek, class, atau apa pun itu antar modul. Untuk melakukannya, Anda perlu mengekspor nilai pada module tersebut.

Untuk mengekspornya, simpanlah nilai tersebut pada properti module.exports. Contoh seperti ini:

* [**coffee.js**](https://www.dicoding.com/academies/261/tutorials/14217#tab1-code1)

1. const coffee = {
2. name: 'Tubruk',
3. price: 15000,
4. }

7. module.exports = coffee;

Setelah itu nilai coffee dapat digunakan pada berkas JavaScript lain dengan cara mengimpor nilainya melalui fungsi global require().

* [**app.js**](https://www.dicoding.com/academies/261/tutorials/14217#tab2-code1)

1. const coffee = require('./coffee');
3. console.log(coffee);
5. /\*\*
6. \* node app.js
7. \*
8. \* output:
9. \* { name: 'Tubruk', price: 15000 }
10. \*/

Perhatikan nilai parameter yang diberikan pada require(). Parameter merupakan lokasi dari module target impor. Ingat! Jika Anda hendak mengimpor modul lokal (*local module*), selalu gunakan tanda ./ di awal alamatnya ya.

Bila berkas **coffee.js** diletakkan di folder yang berbeda dengan **app.js**, contohnya memiliki struktur seperti ini:

1. root folder:.
2. ├── app.js
3. ├── package.json
4. └── lib
5. └── coffee.js

Maka kita perlu mengimpornya dengan alamat:

* [**app.js**](https://www.dicoding.com/academies/261/tutorials/14217#tab3-code1)

1. const coffee = require('./lib/coffee');

Anda juga bisa menggunakan tanda ../ untuk keluar dari satu level folder. Ini berguna bila ingin mengimpor module yang berbeda hirarki seperti ini:

* [**app.js**](https://www.dicoding.com/academies/261/tutorials/14217#tab4-code1)

1. const coffee = require('../lib/coffee');

Bila Anda menggunakan VSCode, Anda akan terbantu dengan fitur auto import yang disediakan. Melalui fitur tersebut Anda tidak perlu repot-repot menuliskan alamat modul secara manual. Tinggal tulis saja nilai yang Anda ingin impor, VSCode akan menangani penulisan fungsi require().

Dalam melakukan impor dan ekspor nilai, kita bisa memanfaatkan object literal dan object destructuring agar dapat mengimpor dan mengekspor lebih dari satu nilai pada sebuah modul. Contoh:

* [**user.js**](https://www.dicoding.com/academies/261/tutorials/14217#tab5-code1)
* [app.js](https://www.dicoding.com/academies/261/tutorials/14217#tab5-code2)

1. const firstName = 'Harry';
2. const lastName = 'Potter';

5. /\* gunakan object literal
6. untuk mengekspor lebih dari satu nilai. \*/
7. module.exports = { firstName, lastName };

Untuk memudahkan developer dalam proses pengembangan, Node.js menyediakan beberapa modul bawaan yang dapat Anda manfaatkan guna mendukung efisiensi untuk melakukan hal-hal yang umum. Modul bawaan tersebut dikenal sebagai core modules. Anda bisa mengimpor *core modules* dengan fungsi yang sama, yakni require().

1. // Mengimpor core module http
2. const http = require('http');

Lokasi core module dituliskan tidak seperti local module. Lokasi bersifat mutlak (core module disimpan folder **lib** pada lokasi Node.js dipasang) sehingga kita cukup menuliskan nama modulnya saja.

Ada 3 jenis modul pada Node.js, Anda sudah mengetahui dua di antaranya. Berikut rinciannya:

* **local module** : module yang dibuat secara lokal berlokasi pada Node.js project Anda.
* **core module** : module bawaan Node.js berlokasi di folder **lib** di mana Node.js terpasang pada komputer Anda. Core module dapat digunakan di mana saja.
* **third party module** : module yang dipasang melalui Node Package Manager. Bila third party module dipasang secara lokal, maka modul akan disimpan pada folder **node\_modules** di Node.js project Anda. Bila dipasang secara global, ia akan disimpan pada folder **node\_modules** di lokasi Node.js dipasang.

Itulah tadi pembahasan mengenai modularisasi. Pada materi selanjutnya kita akan berkenalan dengan third party module dan Node Package Manager.

**Latihan: Modularization**

Sekarang Anda sudah tahu bagaimana cara menerapkan modularisasi pada JavaScript. Namun rasanya tidak afdal bila Anda tidak mempraktikannya sendiri. Untuk menguji pemahaman Anda tentang modularisasi, silakan lakukan latihan berikut.

Buat folder baru dengan nama **modularization** pada proyek nodejs-basic dan di dalamnya buat tiga berkas JavaScript baru yakni **Tiger.js**, **Wolf.js**, dan **index.js**.

Di dalam masing-masing berkas JavaScript, tuliskan starter code berikut:

* [**Tiger.js**](https://www.dicoding.com/academies/261/tutorials/14217#tab6-code1)
* [Wolf.js](https://www.dicoding.com/academies/261/tutorials/14217#tab6-code2)
* [index.js](https://www.dicoding.com/academies/261/tutorials/14217#tab6-code3)

1. class Tiger {
2. constructor() {
3. this.strength = Math.floor(Math.random() \* 100);
4. }
6. growl() {
7. console.log('grrrrr!')
8. }
9. }
11. // TODO 1

Selesaikan kode yang ditandai TODO dengan ketentuan berikut:

* **TODO 1** : Ekspor class Tiger agar dapat digunakan pada berkas JavaScript lain.
* **TODO 2** : Ekspor class Wolf agar dapat digunakan pada berkas JavaScript lain.
* **TODO 3** : Import class Tiger dari berkas **Tiger.js**.
* **TODO 4**: Import class Wolf dari berkas **Wolf.js**.

Setelah selesai mengerjakan TODO, eksekusi berkas index.js dengan perintah:

1. node ./modularization/index.js

Maka console akan menghasilkan output seperti ini:

Grrrr! Harimau memenangkan pertandingan!

**Mengalami kesulitan dalam menyelesaikan latihan?**  
Cobalah untuk ulas kembali materi yang diberikan atau tanyakan kesulitan yang Anda alami pada [forum diskusi](https://www.dicoding.com/academies/261/discussions). Hindari melihat atau membandingkan [kode solusi pada latihan modularization](https://github.com/dicodingacademy/a261-backend-pemula-labs/tree/201-node-js-basic/node-js-basic/modularization" \t "_blank). sebelum Anda mencobanya sendiri.

**Filesystem**

Seluruh data di komputer dikelola dan diakses melalui *filesystem*. Ketika menjalankan kode JavaScript pada browser, sangat penting untuk melimitasi JavaScript dalam mengakses filesystem. Teknik ini dinamakan dengan *sandboxing*. Sandboxing melindungi kita dari program jahat serta tindakan pencurian yang dapat merampas privasi penggunanya.

Bagaimana dengan JavaScript yang dijalankan di back-end? Limitasi tentu tetap ada, namun tidak seketat ketika JavaScript dieksekusi pada browser. Di back-end malah filesystem menjadi fitur esensial karena dalam pengembangan back-end akan sering sekali mengakses atau menulis sebuah berkas di dalam komputer.

Node.js menyediakan core modules fs yang dapat mempermudah kita dalam mengakses filesystem. Setiap method yang ada di module fs tersedia dalam dua versi, yakni versi asynchronous (default) dan versi synchronous.

Tentu Anda sudah tahu apa itu asynchronous dan synchronous kan? Jika belum, pelajari kedua hal tersebut pada kelas [Belajar Dasar Pemrograman JavaScript](https://www.dicoding.com/academies/256).

Untuk mengakses berkas pada komputer kita dapat menggunakan method fs.readFile(). Method ini menerima tiga argumen yakni: lokasi berkas, encoding, dan callback function yang akan terpanggil bila berkas berhasil/gagal diakses.

1. const fs = require('fs');
3. const fileReadCallback = (error, data) => {
4. if(error) {
5. console.log('Gagal membaca berkas');
6. return;
7. }
8. console.log(data);
9. };
11. fs.readFile('todo.txt', 'UTF-8', fileReadCallback);

Sebagai alternatif, Anda juga bisa gunakan method versi synchronous fs.readFileSync().

1. const fs = require('fs');
3. const data = fs.readFileSync('todo.txt', 'UTF-8');
4. console.log(data);

**Latihan: Filesystem**

Pada latihan kali ini, Anda akan ditugaskan untuk membuat program JavaScript yang dapat membaca teks dari berkas **.txt**. Karena Anda sudah belajar filesystem, tentu ini tidak akan sulit.

Silakan buat folder baru dengan nama filesystem. Di dalamnya buat dua berkas, yakni **index.js** dan **notes.txt**.

Di dalam masing-masing berkas, tuliskan kode/teks berikut:

* [**notes.txt**](https://www.dicoding.com/academies/261/tutorials/14642?from=14227#tab1-code1)
* [index.js](https://www.dicoding.com/academies/261/tutorials/14642?from=14227#tab1-code2)

1. Di hari minggu saya akan:
2. 1. Berolahraga pagi.
3. 2. Membersihkan halaman rumah.
4. 3. Menonton film.
5. 4. Membaca buku Laskar Pelangi.

Tugas Anda tertulis jelas seperti yang ada pada berkas index.js. Cobalah tampilkan teks pada berkas notes.txt pada console menggunakan filesystem. Jika TODO sudah selesai dikerjakan, eksekusi berkas index.js dengan perintah:

1. node ./filesystem/index.js

Bila TODO berhasil Anda kerjakan, maka outputnya tampak seperti ini:

**Tips**: Anda bisa gunakan method

1. path.resolve(\_\_dirname, 'notes.txt');

Dari core modules path dalam menetapkan alamat berkas secara lengkap dan dinamis.

**Mengalami kesulitan dalam menyelesaikan latihan?**  
Cobalah untuk ulas kembali materi yang diberikan atau tanyakan kesulitan yang Anda alami pada [forum diskusi](https://www.dicoding.com/academies/261/discussions). Hindari melihat atau membandingkan [kode solusi pada latihan filesystem](https://github.com/dicodingacademy/a261-backend-pemula-labs/tree/201-node-js-basic/node-js-basic/filesystem" \t "_blank). sebelum Anda mencobanya sendiri.

**Readable Stream**

Pada materi sebelumnya Anda sudah mengetahui cara mengakses berkas melalui fungsi fs.readFile(). Fungsi readFile baik versi asynchronous ataupun synchronous, bekerja dengan membaca berkas hingga selesai sebelum mengembalikan data. Itu berarti bila Anda menggunakannya untuk mengakses berkas yang besar, maka akan membutuhkan waktu lama dan memori yang besar untuk mendapatkan hasilnya. Hal ini sungguh tidak efektif.

Solusinya adalah dengan menggunakan teknik stream. Teknik ini tidak membaca berkas secara sekaligus, tapi dengan mengirim bagian demi bagian. Cara inilah yang digunakan oleh YouTube agar video dapat ditampilkan seketika kepada pengguna.

Teknik stream merupakan salah satu konsep fundamental yang mendukung aplikasi Node.js bekerja. Teknik ini dapat menangani kasus baca tulis berkas, komunikasi jaringan, atau beban kerja apapun agar dapat berjalan dengan lebih efisien. Sabar dulu yah, kasus yang disebutkan tadi terlalu kompleks untuk kita pelajari sekarang. Untuk memahami bagaimana stream bekerja kita gunakan kasus paling sederhana, yakni membaca teks pada berkas secara bagian-per-bagian.

Kita dapat membuat readable stream dengan menggunakan method createReadStream() dari core module fs.

1. const fs = require('fs');
3. const readableStream = fs.createReadStream('./article.txt', {
4. highWaterMark: 10
5. });
7. readableStream.on('readable', () => {
8. try {
9. process.stdout.write(`[${readableStream.read()}]`);
10. } catch(error) {
11. // catch the error when the chunk cannot be read.
12. }
13. });
15. readableStream.on('end', () => {
16. console.log('Done');
17. });

Fungsi createReadStream() menerima dua argumen. Yang pertama adalah lokasi berkas yang hendak dibaca, dan yang kedua adalah objek konfigurasi. Di dalam objek konfigurasi kita bisa menetapkan ukuran buffer melalui properti highWaterMark. Nilai default dari properti ini adalah 16384 bytes (16kb). Anda tidak perlu menetapkan properti ini bila ingin tetap memiliki nilai default. Namun karena kita hanya menggunakan berkas teks yang ukurannya sangat kecil, jadi kita buat ukuran buffer menjadi 10 bytes. Itu artinya berkas akan dibaca setiap 10 karakter (1 karakter = 1 bytes).

Buffer di dalam stream adalah memori sementara yang digunakan oleh stream dalam menyimpan data hingga data tersebut dikonsumsi.

createReadStream() mengembalikan EventEmitter, di mana kita dapat menetapkan fungsi listener setiap kali event readable dibangkitkan. Event readable akan dibangkitkan ketika buffer sudah memiliki ukuran sesuai dengan nilai yang ditetapkan pada properti highWaterMark, dalam arti buffer sudah siap dibaca. Kemudian event end akan dibangkitkan setelah proses stream selesai.

Buatlah berkas **article.txt** dan tuliskan teks berikut ini.

* [**article.txt**](https://www.dicoding.com/academies/261/tutorials/14647?from=14642#tab1-code1)

1. Stream di Node.js
3. Teknik stream merupakan salah satu konsep fundamental yang mendukung aplikasi Node.js bekerja. Teknik ini dapat menangani kasus baca tulis berkas, komunikasi jaringan, atau beban kerja apapun agar dapat berjalan dengan lebih efisien.

Bila kode di atas dijalankan maka akan menghasilkan *output* seperti ini:

* [**result**](https://www.dicoding.com/academies/261/tutorials/14647?from=14642#tab2-code1)

[Stream di ][Node.js  
  
][Teknik str][eam merupa][kan salah ][satu konse][p fundamen][tal yang m][endukung a][plikasi No][deJS beker][ja. Teknik][ ini dapat][ menangani][ kasus bac][a tulis be][rkas, komu][nikasi jar][ingan, ata][u beban ke][rja apapun][ agar dapa][t berjalan][ dengan le][bih efisie][n.][null]Done

**Rangkuman Dasar-Dasar Node.js untuk Back-End**

Anda berada di akhir dari modul Dasar-Dasar Node.js untuk Back-End. Mari kita uraikan materi yang sudah Anda pelajari untuk mempertajam pemahaman.

**Pengenalan Node.js**

Dari dulu hingga kini, browser menjadi tempat satu-satunya yang dapat mengeksekusi kode JavaScript. Karenanya, *Web Developer* perlu mempelajari bahasa pemrograman yang berbeda untuk mengembangkan aplikasi *Front-End* dan *Back-End*. Meskipun secanggih dan sekuat apa pun JavaScript berkembang, ia hanya akan digunakan di sisi *Front-End* saja.

Sebenarnya banyak *developer* yang mencoba membuat teknologi agar JavaScript dapat dijalankan di luar *browser*. Namun, belum ada yang berhasil. Hingga pada tahun 2009, Ryan Dahl berhasil menciptakan [Node.js](https://nodejs.org/), teknologi yang diharapkan oleh banyak *web developer*. Tak disangka, saat ini teknologi yang diciptakannya menuai popularitas tinggi. Node.js banyak digunakan oleh perusahaan besar sekelas Netflix, Uber, Paypal, dan eBay.

Node.js berhasil menjadi JavaScript Runtime yang dapat mengeksekusi kode JavaScript di luar *browser*.

**Menjalankan JavaScript Menggunakan Node.js**

Terdapat dua cara dalam menjalankan kode JavaScript menggunakan Node.js, yakni Node.js REPL dan berkas javascript.

* **Node.js REPL**  
  Node.js memiliki fitur REPL atau Read-Eval-Print Loop. Sesuai namanya, fitur ini berfungsi untuk membaca kode JavaScript, mengevaluasi kode tersebut, kemudian mencetak hasil evaluasinya ke console. Nah, untuk *loop*, berarti proses tersebut selalu berulang.  
    
  Jika Anda mengeksekusi console.log(‘Hello NodeJS REPL’), selain pesan “Hello NodeJS REPL”, nilai undefined juga akan tercetak. Hal tersebut terjadi karena REPL selalu menampilkan nilai evaluasi pada console. Karena method console.log() tidak mengembalikan nilai, teks undefined-lah yang tercetak pada console.
* **Berkas JavaScript**  
  Cara lain untuk mengeksekusi kode JavaScript menggunakan Node.js adalah melalui berkas berekstensi .js. Caranya mudah, tinggal buat berkas JavaScript pada proyek Node.js, tulis kode JavaScript di berkas tersebut, lalu eksekusi dengan perintah node namafile.js.

**Node.js Global Object**

Dilansir dari website Node.js, sebenarnya mereka hanya menambahkan beberapa objek saja. Objek tersebut dinamakan dengan ‘true globals’. Berikut adalah daftarnya:

* **global**: Global namespace. Member apa pun yang ada di dalam object ini dapat diakses pada cakupan global.
* **process**: menyediakan interaksi dengan proses Node.js yang berjalan.
* **console**: menyediakan berbagai fungsionalitas [STDIO](http://www.cplusplus.com/reference/cstdio/).
* **setTimeout**, **clearTimeout**, **setInterval**, **clearInterval**: berkaitan dengan waktu.

Ada juga objek yang merupakan ‘pseudo-globals’ atau objek global semu. Objek ini tidak terlihat bila dicetak menggunakan Object.getOwnPropertyName(global) sebab ia bukan member langsung dari objek global, melainkan diturunkan dari cakupan module. Karena pada Node.js semua berkas JavaScript adalah module [[3]](https://javascript.info/modules-intro), objek pseudo-globals dapat diakses layaknya global objek. Berikut adalah daftarnya:

* **module**: digunakan untuk sistem modularisasi pada Node.js.
* **\_\_filename**: keyword untuk mendapatkan lokasi berkas JavaScript yang dieksekusi. Keyword ini tidak tersedia pada Node.js REPL.
* **\_\_dirname**: keyword untuk mendapatkan root directory dari berkas JavaScript yang dieksekusi.
* **require**: digunakan untuk mengimpor module JavaScript.

**Process Object**

Pada Node.js, global objek process memiliki fungsi dan properti yang dapat memberikan informasi mengenai proses yang sedang berjalan.

Salah satu yang sering digunakan adalah properti process.env. Melalui properti ini, kita dapat menyimpan nilai atau mendapatkan informasi mengenai *environment* yang digunakan selama proses sedang berlangsung. Contoh, process.env memiliki properti process.env.PWD yang menyediakan informasi mengenai lokasi di mana proses dijalankan; properti process.env.USER menyimpan informasi nama user pada komputer Anda; dan masih banyak properti lainnya. Anda bisa lihat daftar lengkap properti yang ada pada halaman [dokumentasi Node.js mengenai process.env](https://nodejs.org/dist/latest-v8.x/docs/api/process.html" \l "process_process_env" \t "_blank).

Anda juga bisa secara manual menyimpan nilai di dalam process.env. Hal ini berguna untuk menentukan alur code seperti if-else dalam program berdasarkan *environment* yang Anda berikan. Contohnya, ketika Anda ingin nilai variabel host berbeda di kala pengembangan (*development*) dan produksi (*production*), Anda bisa membuat properti NODE\_ENV pada process.env. Jadi, Anda bisa menentukan nilai host berdasarkan kondisi NODE\_ENV.

**Modularization**

Makin kompleks program yang dikembangkan, Makin kompleks pula kode yang dituliskan. Jika kode aplikasi hanya ditulis dalam satu berkas saja, tentu itu akan menyulitkan kita atau developer lain untuk membaca dan memelihara aplikasi. Idealnya, satu berkas JavaScript hanya memiliki satu tanggung jawab saja. Bila lebih dari satu, itu berarti Anda perlu berkenalan dengan modularization atau modularisasi.

Modularisasi dalam pemrograman merupakan teknik pemisahan kode menjadi modul-modul yang bersifat independen, tetapi bisa saling digunakan untuk membentuk suatu program yang kompleks. Pemisahan kode menjadi modul-modul terpisah inilah yang dapat membuat kode JavaScript lebih mudah dikelola.

Pada Node.js, setiap berkas JavaScript adalah module. Anda bisa membagikan nilai variabel, objek, class, atau apa pun antar module. Untuk melakukannya, Anda perlu mengekspor nilai pada module tersebut.

Untuk mengekspornya, simpanlah nilai tersebut pada properti module.exports.

**Node Package Manager**

Dalam pengembangan aplikasi saat ini, industri gencar memanfaatkan module atau package luar agar proses pengembangan dapat lebih cepat. Semakin kompleks aplikasi tersebut, semakin banyak pula module/package yang digunakan. Di sinilah kita memerlukan sebuah package Manager.

Node Package Manager (NPM) merupakan pengelola package untuk JavaScript yang dapat memudahkan kita dalam mengelola package yang tersedia pada <https://www.npmjs.com/>. NPM merupakan standard package manager yang disediakan oleh Node.js dan sudah otomatis terpasang ketika memasang Node.js pada komputer kita. NPM dapat dioperasikan melalui CMD atau Terminal, Anda pun sudah mencobanya ketika membuat proyek JavaScript.

Selain untuk membuat proyek JavaScript, NPM dapat digunakan untuk memasang atau menghapus *third party module* (modul pihak ketiga). Modul yang dipasang melalui NPM akan disimpan pada folder **node\_modules**.

Terdapat dua tipe pemasangan module melalui NPM: global dan lokal.

* Bila dipasang secara global, module akan bersifat layaknya core module dan dapat digunakan di mana pun.
* Bila dipasang secara lokal, module hanya dapat digunakan pada cakupan proyek Node.js saja.

Namun, saat ini kami sangat menyarankan Anda untuk memasang modul pihak ketiga secara lokal saja. Hindari pemasangan modul secara global karena akan menyebabkan banyak masalah. Sebaiknya gunakan *npx* bila Anda ingin menjalankan Node.js package di mana pun yang Anda inginkan.

**Events**

Aplikasi Node.js biasanya dikenal memiliki pola *event-driven* atau memiliki alur berdasarkan suatu kejadian. Apa maksudnya itu? Mari kita jelajahi lebih dalam lagi.

Dunia nyata penuh dengan kejadian. Alarm berbunyi, ponsel berdering, turun hujan, ataupun kejadian lainnya. Sebagai manusia, kita membuat keputusan, lantas bertindak berdasarkan kejadian yang ada. Contohnya:

* Ketika berjalan dan tetiba turun hujan, kita bergegas menggunakan payung.
* Ketika ponsel berdering, kita bereaksi dengan mengangkat panggilan.
* Ketika merasa lapar, kita makan.

Seperti inilah pola yang terjadi di kehidupan nyata, sudah sejak lama kita bertahan hidup dengan pola seperti ini. Inilah yang dimaksud dengan pola *event-driven*.

Kita kembali ke dunia komputer. Tradisionalnya, programming dilakukan dengan cara yang imperatif. Agar komputer dapat melakukan sesuatu hal, kita perlu banyak menuliskan instruksi secara runtut beserta langkah-langkahnya. Komputer akan membaca kode dari atas ke bawah sesuai dengan urutan yang kita definisikan.

Dengan pola yang kaku seperti itu, kita akan sulit membangun program yang dapat menangani suatu kejadian. Karena kita saja tidak tahu kapan suatu kejadian akan terjadi, lantas bagaimana cara memberikan instruksi pada komputer? Lalu, bagaimana solusinya? Berkaca dari dunia nyata, program komputer juga harus bekerja dengan pola *event-driven*. Syukurlah dengan Node.js kita dapat menerapkan pola tersebut dengan mudah.

Node.js menyediakan EventEmitter class yang merupakan member dari events core module.

**Filesystem**

Ketahuilah bahwa seluruh data di komputer dikelola dan diakses melalui *filesystem*. Ketika kita menjalankan kode JavaScript pada browser, sangat penting untuk melimitasi JavaScript dalam mengakses filesystem. Teknik ini dinamakan dengan *sandboxing*. Sandboxing melindungi kita dari program jahat serta tindakan pencurian yang dapat merampas privasi penggunanya.

Bagaimana dengan JavaScript yang dijalankan di back-end? Limitasi tentu tetap ada, tetapi tidak seketat ketika JavaScript dieksekusi pada browser. Di back-end, justru filesystem menjadi fitur esensial karena dalam pengembangan back-end akan sering sekali mengakses atau menulis sebuah berkas di dalam komputer.

Node.js menyediakan core modules fs yang dapat mempermudah kita dalam mengakses filesystem. Setiap method yang ada di module fs tersedia dalam dua versi, yakni versi asynchronous (default) dan versi synchronous.

Tentu Anda sudah tahu apa itu asynchronous dan synchronous 'kan? Jika belum, pelajari kedua hal tersebut pada kelas [Belajar Dasar Pemrograman JavaScript](https://www.dicoding.com/academies/256).

Untuk mengakses berkas pada komputer, kita dapat menggunakan method fs.readFile(). Method ini menerima tiga argumen: lokasi berkas, encoding, dan callback function yang akan terpanggil bila berkas berhasil/gagal diakses.

**Readable Stream**

Fungsi readFile, baik versi asynchronous maupun synchronous, bekerja dengan membaca berkas hingga selesai sebelum mengembalikan data. Itu berarti, bila Anda menggunakannya untuk mengakses berkas yang besar, proses tersebut akan membutuhkan waktu lama dan memori yang besar untuk mendapatkan hasilnya. Hal ini sungguh tidak efektif!

Lantas, bagaimana? Solusinya adalah dengan menggunakan teknik stream. Teknik ini tidak membaca berkas secara sekaligus, melainkan dengan mengirim bagian demi bagian. Cara inilah yang digunakan oleh YouTube agar video dapat ditampilkan seketika kepada pengguna.

Teknik stream merupakan salah satu konsep fundamental yang mendukung aplikasi Node.js bekerja. Teknik ini dapat menangani kasus baca tulis berkas, komunikasi jaringan, atau beban kerja apa pun agar dapat berjalan dengan lebih efisien. Sabar dulu ya, kasus yang disebutkan tadi terlalu kompleks untuk kita pelajari sekarang. Untuk memahami bagaimana stream bekerja, kita akan gunakan kasus paling sederhana, yakni membaca teks pada berkas secara bagian-per-bagian.

Kita dapat membuat readable stream dengan menggunakan method createReadStream() dari core module fs.

**Writable Stream**

Apakah Anda tahu bahwa teknik *stream* juga dapat digunakan untuk menulis berkas? Teknik ini disebut *writable stream*. Untuk membuat *writable* *stream* dalam menulis berkas, gunakanlah method createWriteStream() dari core module fs.

Fungsi ini menerima satu argumen yakni alamat berkas untuk menyimpan hasil data yang dituliskan. Berkas *output* akan dibuat secara otomatis jika tidak ada. Namun, bila berkas tersebut sudah ada, data sebelumnya akan tertimpa!

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!

**Video Dasar-Dasar Node.js**

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

**Membangun Web Service menggunakan Node.js**

Selamat! Anda sudah belajar dasar dari Node.js. Kini, Anda telah memiliki bekal cukup untuk melangkah ke materi selanjutnya.

Seperti yang sudah Anda pahami, karena hadirnya Node.js, JavaScript tak lagi dikekang oleh browser. Node.js membuat JavaScript bebas dijalankan di mana saja. JavaScript menjadi bahasa pemrograman server side layaknya Java, PHP, C#, atau bahasa lainnya. Sekarang Anda bisa membuat apa saja menggunakan JavaScript! Termasuk aplikasi yang akan kita buat saat ini.

Yeay! Pada modul ini kita akan membuat web service menggunakan JavaScript, hal yang sulit dilakukan tanpa adanya Node.js. Karena itu, pastikan Anda mengikuti materi ini dengan baik yah! Hingga pada akhir modul ini Anda dapat:

* Membangun Web Service menggunakan Node.js secara *native*.
* Mengenal Node.js HTTP framework.
* Membangun Web Service menggunakan Hapi.js (Node.js HTTP framework).
* Membangun RESTful API sederhana.

Sudah tidak sabar ingin memulai membuat web service? yuk lanjut ke materi selanjutnya yah.