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 modularisas

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

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

```
}


module.exports = coffee;
```

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

app.js

```
const coffee = require('./coffee');

console.log(coffee);

/**
 * node app.js
 *
 * output:
 * { name: 'Tubruk', price: 15000 }
 */
```

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:

```
root folder:.
├── app.js
├── package.json
└── lib
    └── coffee.js
```

Maka kita perlu mengimpornya dengan alamat:

app.js

```
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

```
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 app.js

```
1.  const firstName = 'Harry';
2.  const lastName = 'Potter';
3.
4.
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.

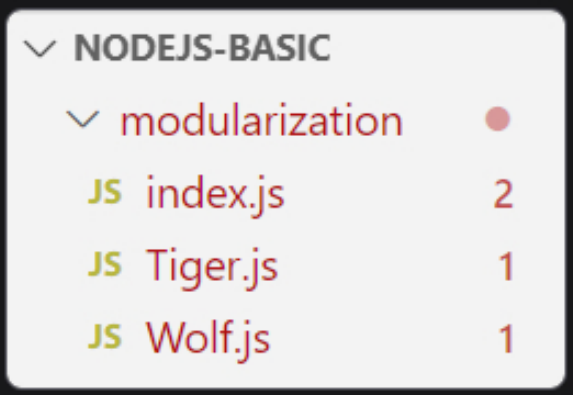
Itulah tadi pembahasan mengenai modularisasi. Pada materi selanjutnya kita akan berkenalan dengan third party module dan Node Package Manager.

Latihan: Modularization

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 Wolf.js index.js

```
1. class Tiger {
2.   constructor() {
3.     this.strength = Math.floor(Math.random() * 100);
4.   }
5.
6.   growl() {
7.     console.log('grrrrr!')
8.   }
```

```
}

// 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:

```
node ./modularization/index.js
```

Maka console akan menghasilkan output seperti ini:

```
PROBLEMS 4  OUTPUT  DEBUG CONSOLE  TERMINAL

C:\javascript-projects\nodejs-basic>node ./modularization/index.js
grrrrr!


C:\javascript-projects\nodejs-basic>
```

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. Hindari melihat atau membandingkan kode solusi pada latihan modularization. sebelum Anda mencobanya sendiri.