FULL STACK GATE

devsaurus

Full Stack Gate

Sebuah pendahuluan untuk memulai karir sebagai seorang full stack web developer.

TABLE OF CONTENT

Daftar Isi

- Do While
- Break
- Continue
- Function
  - Return

HTML & CSS
- HTML
  - HTML element
  - HTML attribute
  - HTML Display
  - HTML id & class
- CSS
  - Menambahkan CSS ke HTML.
  - 7 konsep Dasar CSS
  - Konsep Dasar lainnya
  - CSS Framework dan Extension
    - SASS vs SCSS vs LESS

JavaScript
- 1. Konsep Dasar JavaScript
  - ECMAScript
  - JavaScript Engine
    - Dynamic vs Static
  - JavaScript Pada Browser
    - BOM & DOM
    - Menambahkan JS ke HTML
    - Event
  - Variable
    - Lexical Environment
  - Type Conversions
  - String
    - BackTicks
    - Substring
    - Replace
  - Regular Expression (Regex)
  - Object
    - Cara Membuat Object
    - Manipulasi Object
    - Method
  - Array

- React element
- React Component
- Membuat Component
  - Hanya ada satu root component
  - Satu component hanya menghasilkan satu element
- Data Handling
- Props
- State
  - Inisialisasi state
  - Mengakses value dari state
  - Mengubah state
  - Pilih mana antara props dan state?
- Komunikasi data antar component
- Event Handling
- Lifecycle Methods
- Styling
- Conditional Rendering & List
- Conditional Rendering
- List
- Key
- Form
- Controlled Components
- Membuat Form
- Uncontrolled Components
  - Refs
- Hooks
- React Class Component vs React Function Component
- React Class Component
  - Membuat Component
  - Data Handling
  - Lifecycle Method
- React Function Component
  - Membuat Component
  - Data Handling
  - Lifecycle Methods
- Ekosistem

Node.js

- Arsitektur dan Cara Kerja Node.js
- Membuat aplikasi server

- Install Node.js
- Module
- NPM
- Membuat simple HTTP Server

Membuat REST API

- REST
- HTTP Method

Express.js

- Setup
- Routing
- Middleware
- Static Files
- Error Handling
  - Handle Asynchronous Error
  - Membuat Error Handling Middleware

Database

- Relational Database
- Non-relational Database

MySQL

- Cara Install
- Testing MySQL
- Operasi Dasar pada Database
- Operasi Dasar pada Tabel
- Manipulasi Data
- Menampilkan data dari lebih dari satu tabel
- Manipulasi Data (Lanjutan)
- Table Relationship / Relasi antar tabel

MongoDB

- NoSQL vs SQL
  - Bentuk data
  - Read Operation
  - Scalability
  - Install
- Operasi Dasar MongoDB
  - Operasi Dasar pada Database
  - Operasi Dasar pada Collection
  - CRUD (Create, Read, Update, Delete) Operation
  - Aggregation

- Database GUI / Tools

Deployment
- Deployment Environment
- Cloud Computing
  - Cloud Computing provider
  - Cloud Computing Service Model
- Cara Deploy
  - Persiapan
  - Amazon Lightsail
  - Heroku
  - Netlify
- Serverless
  - Definisi
  - Kelebihan & Kekurangan
  - Kelebihan
  - Kekurangan
  - Cloud Provider
  - Serverless Computing Services / Solutions / Platform

Tools
- Version Control
  - Jenis Version Control
  - Git
    - Branch
    - The 3 Tree
    - Install Git
    - Operasi dan Perintah Dasar pada Git
    - Git Provider
    - Git Software
- NPM
  - Penggunaan
- Webpack
- Babel
- Browser Developer Tools
- Linter
  - Setup ESLint (VS Code)
  - Prettier + Style Guide
- Testing
  - Testing Tools
- CI/CD

FRONT END

03

JavaScript

Pembahasan JavaScript akan dibagi menjadi 4 bagian

1. Konsep dasar JavaScript
2. Konsep JavaScript sebagai pendahuluan React
3. Teknik Coding JavaScript
4. Fitur-fitur terbaru JavaScript

1. Konsep Dasar JavaScript

ECMAScript

ECMAScript adalah standar yang dikeluarkan oleh badan standarisasi international **ECMA International** untuk scripting language.

Salah satu bahasa pemrograman yang menerapkan standard ECMAScript adalah JavaScript.

Versi dari ECMAScript sendiri cukup banyak dan hampir tiap tahun dirilis.

Tabel versi ECMAScript:

Official Name	Edisi	Tahun Rilis
ECMAScript 1	ES1	1997
ECMAScript 2	ES2	1998
ECMAScript 3	ES3	1999
ECMAScript 4	ES4	-
ECMAScript 5	**ES5**	**2009**
ECMAScript 5.1	ES5.1	2011
ECMAScript 2015	ES6	2015
ECMAScript 2016	ES7	2016

ECMAScript 2017	ES8	2017
ECMAScript 2018	ES9	2018
ECMAScript 2019	ES10	2019
ECMAScript 2020	ES11	2020

Versi ES5 ke atas adalah versi dari ECMAScript yang paling banyak digunakan, karena mulai versi inilah diperkenalkan banyak sekali fitur penting untuk membantu mengembangkan program menggunakan JavaScript.

Dan hampir semua modern browser saat ini support versi ES5 keatas.

Pembahasan JavaScript di buku ini adalah JavaScript versi ES5 keatas.

JavaScript Engine

JavaScript termasuk **Interpreted Language**, yang berarti ada program lain yang bertugas untuk menerjemahkan (translate) JavaScript code ke bahasa mesin dengan kata lain ada program yang bertugas untuk mengeksekusi code JavaScript.

Program itu adalah **JavaScript Engine**, setiap browser memiliki versi JavaScript Engine-nya masing-masing.

Browser Google Chrome menggunakan JavaScript Engine bernama **Chrome V8**
Firefox menggunakan JavaScript Engine bernama **GreaseMonkey**
dll

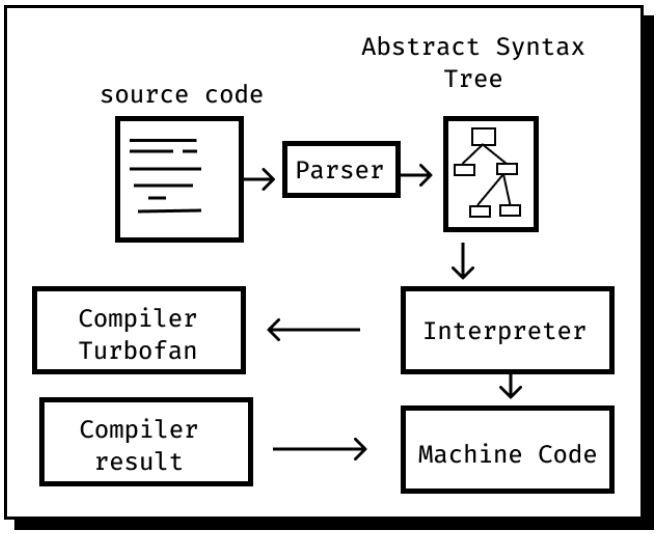
Namun perlu diketahui V8 yang merupakan JavaScript Engine paling populer saat ini menggunakan metode yang berbeda untuk menerjemahkan source code ke machine code, metode yang digunakan adalah **Just-In-Time(JIT) Compilation**.

Just-In-Time(JIT) Compilation merupakan gabungan dari *Compiler* dan *Interpreter*.

Secara sederhana metode ini bekerja sebagaimana interpreter namun dalam prosesnya terdapat proses optimasi, yaitu melakukan *compile* terhadap bagian dari code yang sering dieksekusi kemudian menyimpan hasil compile tersebut.

Sehingga jika code tersebut dieksekusi kembali maka JavaScript Engine tidak akan melakukan proses translate dari awal namun akan menggunakan versi compiled yang sudah disimpan sebelumnya.

Hal ini menjadikan proses translate JavaScript code ke machine code menjadi lebih cepat dan efisien.



Dynamic vs Static

Selain termasuk Interpreted Language, JavaScript juga termasuk **Dynamic-Typed language**.

Dimana semua data ditentukan tipenya oleh interpreter saat *runtime* (ketika aplikasi sedang dijalankan).

Oleh karena itu kita bisa menyimpan value ke dalam suatu variable tanpa harus menentukan tipe dari data tersebut.

```
let dinoName = 'brachio';
```

Seperti contoh code diatas kita tidak menentukan tipe data dari variable **dinoName** apakah itu String, Number atau Object.

Hal ini berbeda dengan Static-Typed language seperti Java atau C#, dimana tipe data harus ditentukan terlebih dahulu.

```
int dinoAge = 1000;
```

Code diatas merupakan contoh deklarasi variable pada bahasa pemrograman Java & C#, tipe data dari variable **dinoAge** adalah **int** (**integer**) dan harus ditentukan di awal.

Hal ini penting untuk diketahui agar kita lebih berhati-hati dalam menulis code JavaScript untuk menghindari bug yang tidak diinginkan karena JavaScript tidak melakukan proses **type-checking**.

Sebagai contoh pada JavaScript kita bisa melakukan operasi penjumlahan seperti ini:

```
let dinoName = 't-rex';
let dinoAge = 1000;

let total = dinoName + dinoAge;
```

Sekilas code diatas terlihat aneh, menambahkan teks dengan angka secara matematika adalah sesuatu yang tidak dilakukan, namun JavaScript akan tetap mengeksekusinya dan kita akan mendapatkan hasil yang tidak terduga.

Sedangkan pada Static-Typed language, jika code diatas dieksekusi maka akan muncul pesan error bahwa operasi penjumlahan tersebut tidak bisa dilanjutkan karena ada perbedaan tipe data antara kedua variable.

Saat ini kita bisa menggunakan TypeScript atau Flow untuk mengatasi masalah type-checking pada JavaScript.

JavaScript Pada Browser

BOM & DOM

BOM dan DOM adalah konsep yang penting untuk diketahui dalam pemrograman JavaScript.

JavaScript menggunakan BOM untuk mengakses isi dari browser dan menggunakan DOM untuk mengakses HTML.

BOM

BOM atau **Browser Object Model** adalah hirarki atau urutan tingkatan dari semua object yang ada di dalam browser.

5 object di dalam BOM:

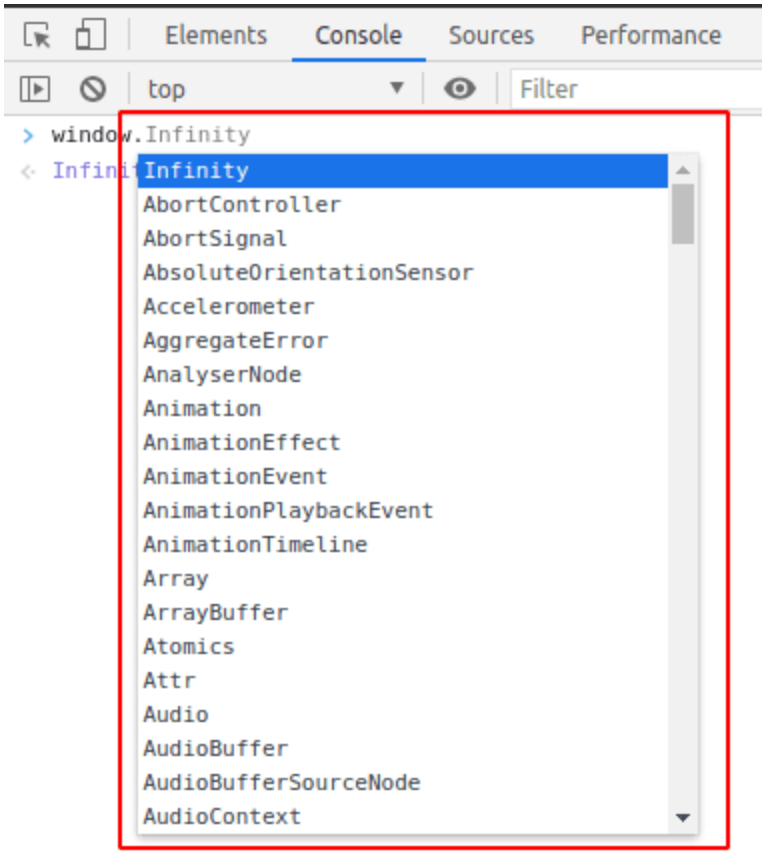
1. Document
2. Location
3. Navigator
4. Screen
5. History

Semua object di atas juga merupakan bagian dari global variable bernama **window** object.

Jadi window object adalah object yang berada di urutan teratas dari BOM.

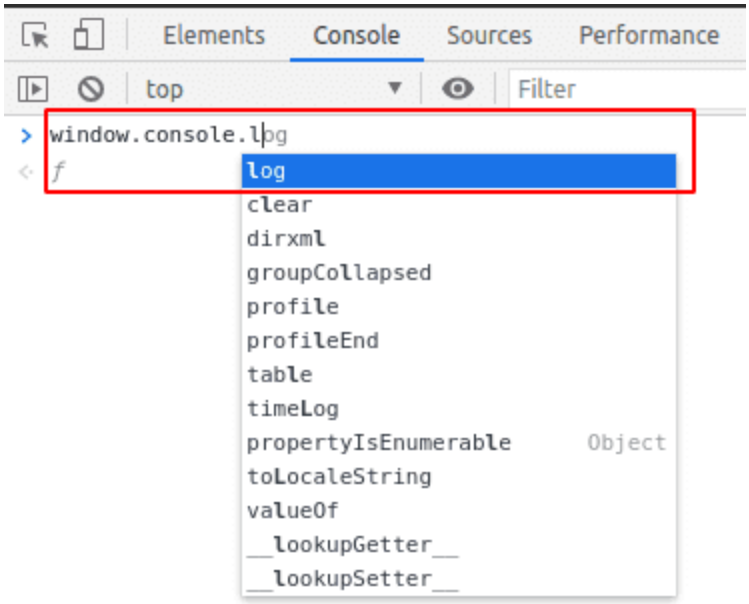
Kamu bisa melihat semua fungsi yang ada didalam browser pada **window** object.

Buka Browser Chrome -> DevTools, kemudian pada tab console ketik **window** maka kamu akan melihat daftar object dan fungsi yang ada di browser tersebut.



Di dalam window object terdapat document object yang berisi DOM.

Selain document object, di dalam window object juga terdapat console object yang berfungsi untuk menampilkan informasi pada console di browser.



Jadi kita tahu bahwa **console.log()** bukan bagian dari JavaScript.

DOM

DOM = Document Object Model

DOM adalah gambaran struktur dari suatu dokumen dalam bentuk **nodes & objects**.

Halaman web adalah suatu jenis dokumen. Sehingga DOM pada web adalah gambaran struktur dari web tersebut.

Jika halaman web tersebut dibuat menggunakan HTML maka disebut HTML DOM.

Dengan HTML DOM, Javascript dapat memodifikasi elemen HTML.

BOM vs DOM

- DOM berkaitan dengan halaman web (HTML atau XML), dan BOM berkaitan dengan browser
- DOM ada di dalam window object, dimana window object adalah bagian dari BOM

Menambahkan JS ke HTML

Ada 2 cara menambahkan JavaScript ke HTML.

1. Embed / Internal
Ditambahkan di antara <script> tag pada satu file yang sama.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Devsaurus</title>
  </head>

  <body>
    <p id="el"></p>

    <script>
      document.getelemenById('el').innerHTML = 'Hello Brachio';
    </script>
  </body>
</html>
```

2. External

Ditambahkan di antara <script> tag pada file yang berbeda.

filename: **script.js**

```
document.getelemenById('el').innerHTML = 'Hello Brachio';
```

filename: **index.html**

```
<!DOCTYPE html>
<html>
  <head>
    <title>Devsaurus</title>
  </head>

  <body>
    <p id="el"></p>
```

```
    <script src="script.js"></script>
  </body>
</html>
```

<script> tag bisa ditaruh di dalam <head> atau <body>

Event

JavaScript pada browser menggunakan *event programming model*, semua operasi dijalankan berdasarkan kejadian atau event tertentu.

Contoh event:

onclick (user click sebuah elemen HTML)
onchange (elemen HTML mengalami perubahan)
onload (browser telah selesai load halaman web)

Perlu diperhatikan bahwa event-event diatas bukan bagian dari JavaScript namun merupakan bagian dari HTML DOM API.

Contoh code untuk click event:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Devsaurus</title>
  </head>
  <body>
    <button onclick="sayHi()">Say Hi</button>

    <p id="el"></p>

    <script>
      function sayHi() {
        document.getelemenById('el').innerHTML = 'Hello Brachio';
      }
    </script>
  </body>
</html>
```

Pada code diatas, browser akan menampilkan tulisan **Hello Brachio** ketika button **Say Hi** di klik.

Variable

Ada 3 cara mendeklarasikan sebuah variable di JavaScript yaitu menggunakan **var, let & const**.

Pada bagian ini kita akan jelaskan perbedaan paling mendasar dari ketiganya yaitu **Lexical Environment**.

Lexical Environment

Ketika sebuah variable dibuat atau dideklarasikan, JavaScript akan membuat dan menyimpan variable tersebut di dalam sebuah lexical environment.

Ada 4 jenis lexical environment.

- Global Environment
- Module Environment
- Function Environment
- Block Environment

Setiap environment ini memiliki cakupan atau scope-nya sendiri-sendiri.

Scope dapat diartikan area dimana variable itu bisa diakses.

Var

Sebuah variable yang dideklarasikan menggunakan keyword var bersifat **globally-scoped** atau **function scoped**.

Artinya jika variable dideklarasikan di dalam sebuah function maka variable itu hanya bisa diakses di dalam function tersebut, jika dideklarasikan di luar function maka variable dapat diakses dimana saja.

Let

Variable yang dideklarasikan dengan keyword let bersifat **block-scoped**, hanya bisa diakses di dalam block ({...}) dimana variable dibuat.

Const

Sama dengan let, hanya saja value atau nilai dari variable tidak bisa diubah atau bersifat read-only

Type Conversions

Kamu bisa mengkonversi dari satu tipe data ke tipe data yang lain, proses konversi ini biasa disebut **type conversions** atau **casting**.

Beberapa type conversions yang ada di JavaScript:

String Conversion

Contoh: mengubah data dari number ke string.

```
let dinoAge = 123;

dinoAge = String(dinoAge);
// atau menggunakan
// dinoAge = dinoAge.toString();

console.log(typeof(dinoAge)); //string
```

Numeric Conversions

Contoh: Mengubah string ke integer.

```
let dinoAge = '123';

dinoAge = Number(dinoAge);
// atau dinoAge = parseInt(dinoAge, 10);

console.log(typeof(dinoAge)); // number
```

Boolean Conversions

Contoh: Mengubah number ke boolean.

```
let dinoAge = 123;

dinoAge = Boolean(dinoAge);
```

```
console.log(typeof(dinoAge)); // boolean
console.log(dinoAge); // true
```

Perlu diperhatikan jika kamu mengubah tipe data berikut ini ke boolean maka hasilnya adalah **false**:

- null
- undefined
- NaN
- "" (double quotes)
- 0

Selain tipe data diatas maka hasilnya adalah **true**.

String

Ada 3 cara membuat string pada JavaScript yaitu:

- single quote (')
- double quote(""), dan
- backticks(``)

Tidak ada perbedaan antara penggunaan **single quote** dan **double quote**, hanya saja disarankan untuk selalu menggunakan salah satu diantara keduanya dan tidak mengkombinasikan keduanya dalam satu file.

BackTicks

Pada bagian ini kita akan membahas penggunaan backticks untuk multiline string dan template literal.

Multiline String

menggunakan backticks untuk multiline string.

```
let multiline = `
    Baris pertama
    Baris Kedua
    -
    Baris Keempat
`;
```

jika kita menggunakan single quote atau double quote maka harus ditulis seperti ini.

```
let multiline = '
    Baris pertama \n \
    Baris Kedua \n \
    - \n \
    Baris Keempat \n \
    ';
```

Template literal

Template literal adalah format string dimana kita bisa menyisipkan *expression* di dalamnya.

```
let dinoName = 'brachio';

let sayHi = `Hi ${dinoName}, How are you?`;
```

Pada code diatas dinoName adalah sebuah expression.

Tanpa backticks code harus ditulis seperti ini:

```
let dinoName = 'brachio';

let sayHi = 'Hi ' + dinoName  + ', How are you?'';
```

Substring

Substring adalah bagian dari string.

Sebagai contoh jika kita ingin mendapatkan kata **dino** pada kalimat **Hi dino, How are you?**, kita bisa menggunakan 3 cara:

1. string.substring(start, end)

Mendapatkan bagian dari string antara **start** dan **end** (end tidak termasuk).

```
let sayHi = 'Hi dino, How are you?';

let substring = sayHi.substring(3, 7);

console.log(substring); // dino
```

2. string.substr(start, length)

Mendapatkan bagian dari string mulai dari **start** sebanyak **length**.

```
let sayHi = 'Hi dino, How are you?';

let substring = sayHi.substr(3, 4);

console.log(substring); // dino
```

substr juga dapat menerima argument negatif, namun hanya untuk start.

```
let sayHi = 'Hi dino, How are you?';

let substring = sayHi.substr(-1, 3);

console.log(substring); // you
```

3. string.slice(start, end)

Mendapatkan bagian dari string antara start sampai karakter sebelum end (tidak termasuk end).

```
let sayHi = 'Hi dino, How are you?';

let substring = sayHi.slice(3, 5);

console.log(substring);
```

Bedanya dengan substring, slice dapat menerima argument negatif.

```
let sayHi = 'Hi dino, How are you?';

let substring = sayHi.slice(-4, -1);

console.log(substring); // you
```

Replace

str.replace() dipakai untuk mengganti sebagian string dengan string yang lain.

```
let sayHi = 'Hi dino, How are you?';

let sayOther = sayHi.replace('dino', 't-rex');

console.log(sayOther); // Hi t-rex, How are you?
```

Regular Expression (Regex)

Regular Expression(Regex) adalah pola yang digunakan sebagai kriteria untuk mendapatkan kombinasi karakter pada suatu string.

Syntax

```
/pattern/flag;
```

pattern = pola yang digunakan

flag = tambahan kriteria

Penggunaan

Contoh menggunakan regex pada **str.replace()**:

```
let word = 'Hello Brachio';
let regexPattern = /Brachio/i;

let newWord = word.replace(regexPattern, 't-rex');

console.log(newWord); // Hello t-rex
```

Pada code diatas kita mencari kata Brachio dengan menggunakan pola regex **/Brachio/i** yang kemudian kita ganti dengan kata **t-rex**.

flags yang ada di JavaScript

flags	Keterangan
g	Global Search
i	Case Sensitive

m	Multiline
s	Allow . character
u	Unicode Support
y	Sticky Search

Special Character

Pola regex umumnya dibuat menggunakan special character.

Sebagai contoh \d atau [0-9] adalah special character yang digunakan untuk mencari angka dalam sebuah string.

```
let word = 'Brachiosaurus has been estimated at 20 meters';
let regexPattern = /[0-9]+/;

let newWord = word.replace(regexPattern, 140);

console.log(newWord); // Brachiosaurus has been estimated at 140 meters
```

Pada code diatas special character [0-9]+ mewakili angka 20.

Daftar lengkap special character yang biasa digunakan untuk membentuk pola regex dapat ditemukan disini.

Jika ingin memahami Regular Expression lebih jauh silahkan lihat disini.

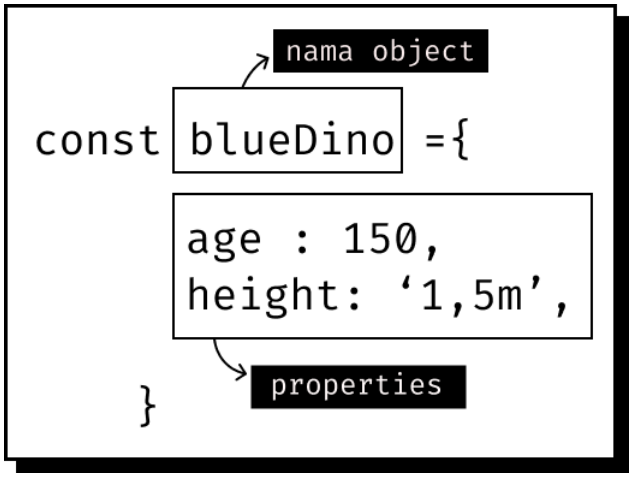
Object

Tipe data pada JavaScript dibagi menjadi dua yaitu **Primitive** dan **Object**.

Primitive mewakili satu tipe data saja, contoh: **Number, Boolean, String.**

Sedangkan Object dapat mewakili banyak tipe data.

Data di dalam object memiliki format *key-value* atau biasa disebut *properties*.



Cara Membuat Object

```
const redDino = {}; //empty object

const blueDino = {
  // initiate non-empty object and properties
  age: 150,
  height: '1.5 m',
  weight: '500 kg'
};
```

Manipulasi Object

Yang dimaksud manipulasi object disini adalah mengakses data, mengubah data, menambahkan properties dan menghapus properties.

Mengakses data

Ada dua cara mengakses suatu data properties di dalam object, menggunakan **dot.notation(.)** atau **bracket notation []** .

1. dot notation (.)

```
const blueDino = {
  age: 150,
  height: '1.5 m',
  weight: '500 kg'
};
console.log(blueDino.age); // 150
```

2. bracket notation [ ]

```
const blueDino = {
  age: 150,
  height: '1.5 m',
  weight: '500 kg'
};
console.log(blueDino['age']); // 150
```

Sedangkan untuk mengakses setiap properties dari sebuah object kita bisa gunakan **for..in** statement.

```
const blueDino = {
  age: 150,
  height: '1.5 m',
  weight: '500 kg'
};

for (const key in blueDino) {
  console.log(`${key}: ${blueDino[key]}`);
}
```

Mengubah data

Selain untuk mengakses data object, dot dan bracket notation juga digunakan untuk mengubah data object.

```
const blueDino = {
  age: 150,
  height: '1.5 m',
  weight: '500 kg'
};

blueDino.age = 200;
blueDino['weight'] = '650 kg';

console.log(blueDino.age); // 200
console.log(blueDino['weight']); // 650 kg
```

Menambah properties

```
const blueDino = {
  age: 150,
  height: '1.5 m',
  weight: '500 kg'
};

blueDino.name = 'brachio'; // add properties

console.log(blueDino.name);
```

Pada code diatas kita menambahkan properties dengan **key = name** dan **value = 'brachio'**.

Sehingga object **blueDino** memiliki 4 properties, yaitu **age**, **height**, **weight** dan **name**.

Menghapus properties

```
const blueDino = {
  age: 150,
  height: '1.5 m',
  weight: '500 kg'
};
```

```
delete blueDino.name; // delete properties

console.log(blueDino.name); // undefined
```

blueDino.name menjadi **undefined** setelah dihapus.

Method

Method adalah function yang menjadi properties pada object.

```
const blueDino = {
  age: 150,
  height: '1.5 m',
  weight: '500 kg',
  sayHi: function () {
    return 'Hi, How are you?';
  }
};

blueDino.sayHi(); // Hi, How are you?
```

Di JavaScript yang termasuk Object adalah **Array** & **Date**

Array

Sebelumnya kita telah membahas array secara singkat.

Pada bagian ini kita akan bahas array lebih jauh.

Ada 4 operasi dasar pada array, yaitu **Reading, Searching, Inserting** dan **Deletion**.

Reading

Untuk mengakses sebuah item array kamu bisa gunakan index.

```
const dinoGroup = ['t-rex', 'brachio', 'tricera'];
console.log(dinoGroup[0]); // t-rex
```

Sedangkan untuk mengakses setiap item array kamu bisa gunakan **for loop**.

Ada dua bentuk for loop.

for

```
const dinoGroup = ['t-rex', 'brachio', 'tricera'];
for (let index = 0; index < array.length; index++) {
  console.log(array[index]);
}
```

for..of

```
const dinoGroup = ['t-rex', 'brachio', 'tricera'];
for (let dino of dinoGroup) {
  console.log(dino);
}
```

Searching

Kamu bisa mencari sebuah item dalam array menggunakan **for / for..of + if statement**.

```
const dinoGroup = ['t-rex', 'brachio', 'tricera'];


for (let dino of dinoGroup) {
  if (dino === 't-rex') {
    console.log('Found!');
  }
}
```

Tetapi ada cara yang lebih simple, yaitu menggunakan method seperti **find** atau **filter**.

find(function)

menampilkan **item pertama** dari hasil pencarian sesuai dengan kriteria yang sudah ditentukan di dalam sebuah function.

```
const numbers = [10, 9, 11, 12, 8, 13];

function getBigNumber(number) {
  if (number > 10) {
    return number;
  }
}

const result = numbers.find(getBigNumber);

console.log(result); // 11
```

filter(function)

Menampilkan hasil pencarian sesuai dengan kriteria yang sudah ditentukan di dalam sebuah function.

Hasil pencarian disimpan dalam array baru.

```
const numbers = [10, 9, 11, 12, 8, 13];

function getBigNumber(number) {
  if (number > 10) {
    return number;
```

```
  }
}

const result = numbers.filter(getBigNumber);

console.log(result); // [ 11, 12, 13 ]
```

Inserting

Untuk menambahkan atau menyisipkan data ke dalam sebuah array kamu bisa gunakan beberapa method berikut ini:

push(item)

Menambah data/item dan diletakan di akhir array.

```
const dinoGroup = ['t-rex', 'brachio', 'tricera'];

dinoGroup.push('allo');

console.log(dinoGroup); // ['t-rex', 'brachio', 'tricera', 'allo']
```

unshift(...items)

Menambah item dan diletakan di awal array kemudian menggeser index item yang lain (dari index 0 ke 1, 1 ke 2 dst).

```
const dinoGroup = ['t-rex', 'brachio', 'tricera'];

dinoGroup.unshift('allo');

console.log(dinoGroup); // [ 'allo', 't-rex', 'brachio', 'tricera' ]
```

Deletion

Untuk menghapus item array kita bisa gunakan beberapa method berikut ini:

pop()

Menghapus item terakhir dari array.

```
const dinoGroup = ['t-rex', 'brachio', 'tricera'];

dinoGroup.pop();

console.log(dinoGroup); // [ 't-rex', 'brachio' ]
```

shift()

Menghapus item awal dari array kemudian menggeser index item yang lain (dari index 1 ke 0, 2 ke 1 dst).

```
const dinoGroup = ['t-rex', 'brachio', 'tricera'];

dinoGroup.shift();

console.log(dinoGroup); // [brachio', 'tricera']
```

Array Modification

Yang dimaksud memodifikasi array disini adalah kombinasi dari operasi dasar yang sudah dijelaskan.

Beberapa method yang bisa kita gunakan untuk memodifikasi array:

splice(pos, deleteCount, ...items)

Kamu dapat menyisipkan, menghapus dan mengubah item array menggunakan splice.

Contoh 1: Menyisipkan item di index 1.

```
const dinoGroup = ['t-rex', 'brachio', 'tricera'];

dinoGroup.splice(1, 0, 'allo');

console.log(dinoGroup); // [ 't-rex', 'allo', 'brachio', 'tricera' ]
```

Contoh 2: Menghapus item di index 1.

```
const dinoGroup = ['t-rex', 'brachio', 'tricera'];

dinoGroup.splice(1, 1);

console.log(dinoGroup); // [ 't-rex', 'tricera' ]
```

Contoh 3: Mengubah item di index 1.

```
const dinoGroup = ['t-rex', 'brachio', 'tricera'];

dinoGroup.splice(1, 1, 'allo');

console.log(dinoGroup); // [ 't-rex', 'allo', 'tricera' ]
```

slice(start, end)

Sama dengan string.slice namun hasilnya adalah array baru.

```
const dinoGroup = ['t-rex', 'brachio', 'tricera', 'allo'];

const newDinoGroup = dinoGroup.slice(1, 2);

console.log(newDinoGroup); // [ 'brachio' ]
```

forEach(function)

Memodifikasi array dengan cara mengeksekusi sebuah function untuk setiap item array.

Pada contoh code dibawah ini kita menampilkan (*reading*) setiap item pada array menggunakan sebuah function.

```
const dinoGroup = ['t-rex', 'brachio', 'tricera'];

dinoGroup.forEach(dino => console.log(dino));
```

Output

```
t-rex
brachio
tricera
```

Beberapa catatan untuk forEach:

- *return value* pada forEach adalah **undefined**
- function pada forEach lebih baik ditulis menggunakan **arrow function**

Untuk mengetahui detailnya silahkan lihat disini.

map(function)

Memodifikasi array dengan cara mengeksekusi sebuah function untuk setiap item array dimana array hasil modifikasinya adalah sebuah array baru.

```
const numbers = [1, 2, 3, 4, 5];

function doubleIt(number) {
  return number * 2;
}

const result = numbers.map(doubleIt);

console.log(result); // [ 2, 4, 6, 8, 10 ]
```

Date

Pada JavaScript, object Date digunakan untuk memanipulasi data waktu seperti tanggal dan jam.

Contoh:

Membuat Date Object

```
const now = new Date();

console.log(now);
```

Menampilkan komponen dari Date

getDate()

Menampilkan tanggal.

```
const dinoEvent = new Date('January 01, 2020 10:15:30');
const dateEvent = dinoEvent.getDate();
```

getMonth()

Mendapatkan dan menampilkan bulan.

```
const dinoEvent = new Date('January 01, 2020 10:15:30');
const monthEvent = dinoEvent.getMonth();
```

Nama bulan ditulis dalam angka, 0 untuk Januari, 1 untuk Februari dst

getFullYear()

Mendapatkan dan menampilkan tahun.

```
const dinoEvent = new Date('January 01, 2020 10:15:30');
```

```
const yearEvent = dinoEvent.getFullYear();
```

getHours(), getMinutes(), getSeconds(), getMilliseconds()

Menampilkan jam, menit, detik dan milidetik.

```
const dinoEvent = new Date('January 01, 2020 10:15:30');
const hour = dinoEvent.getHours();
const minutes = dinoEvent.getMinutes();
const second = dinoEvent.getSeconds();
const miliSecond = dinoEvent.getMilliseconds();
```

Date.parse()

Digunakan untuk mendapatkan nilai milisecond terhitung mulai 1 Januari 1970, 00:00:00 UTC.

```
const milisecond = Date.parse('January 01, 2020 10:15:30');
console.log(milisecond); // 1577873730000
```

Nilai milisecond di atas disebut dengan **unix time** atau **epoch time**, unix time ini digunakan hampir di semua operating system.

JSON

Ketika browser mengirim data ke sebuah server, data tersebut dikirim dalam format tertentu.

Salah satu format data yang banyak digunakan untuk transfer data antara browser dan server adalah JSON.

JSON adalah singkatan dari **JavaScript Object Notation**, pada dasarnya JSON hanya sebuah text yang memiliki format/syntax sebagai berikut.

```
{
  "key" : "value"
}
```

Contoh

```
{
  "name" : "brachio",
  "weight" : 150,
  "height" : 1.5
}
```

Sekilas syntax dari JSON mirip dengan JavaScript Object namun memiliki perbedaan yaitu key-value JSON harus berupa string yang menggunakan double quote.

Perbedaan JSON dan JavaScript Object

JSON

```
{
  "name" : "brachio",
  "weight" : 150
}
```

JavaScript Object

```
{
  name   : "brachio",
  weight : 150
}
```

Convert JSON

Mengubah JavaScript Object ke JSON dan sebaliknya.

Object to JSON menggunakan **JSON.stringify().**

```
const blueDino = {
  age: 150,
  height: '1.5 m',
  weight: '500 kg'
};

const jsonDino = JSON.stringify(blueDino);

console.log(jsonDino); // {"age":150,"height":"1.5 m","weight":"500 kg"}
```

JSON to Object menggunakan **JSON.parse().**

```
const jsonDino = `{
  "age": "150",
  "height": "1.5 m",
  "weight": "500 kg"
}`;

const blueDino = JSON.parse(jsonDino);

console.log(blueDino); // { age: '150', height: '1.5 m', weight: '500 kg' }
```

Tipe Data JSON

Data JSON dapat berupa String, Number, Nested Object, Array, Boolean.

```
{
  "name" : "brachio",
  "weight" : 150,
  "height" : 1.5,
  "address" : {
    "street": "Jalan Lama",
    "block" : "K"
    "number" : 20
    "city" : "Jakarta",
  },
  "hoby" : [ "football", "swimming", "tennis" ],
  "single" : true
}
```

Namun data JSON tidak dapat berupa function atau date.

```
{
  "name" : function() { console.log('Brachio') } // error
}
```

Function

Deklarasi Function

Deklarasi sebuah function pada JavaScript dapat ditulis seperti ini:

```
function updateDinoName(name) {
  name = 't-rex';
  return name;
}

const dinoName = 'brachio';

updateDinoName(dinoName); // call function
```

Namun bentuk deklarasi function pada JavaScript tidak hanya satu, ada beberapa bentuk yang lain:

Function Expression

Pada function expression, function disimpan di dalam sebuah variable.

Syntax

```
const functionName = function (parameter) {
  ...
}
```

Contoh:

```
const updateDinoName = function (name) {
  name = 't-rex';
  return name;
};

let dinoName = 'brachio';

updateDinoName(dinoName); // call function
```

Perbedaan Function Declaration dengan Function Expression

Kamu bisa memanggil function yang dibuat menggunakan Function Declaration meskipun deklarasi dilakukan setelah pemanggilan.

```
let dinoName = 'brachio';

updateDinoName(dinoName); // call function

function updateDinoName(name) {
  name = 't-rex';
  return name;
}
```

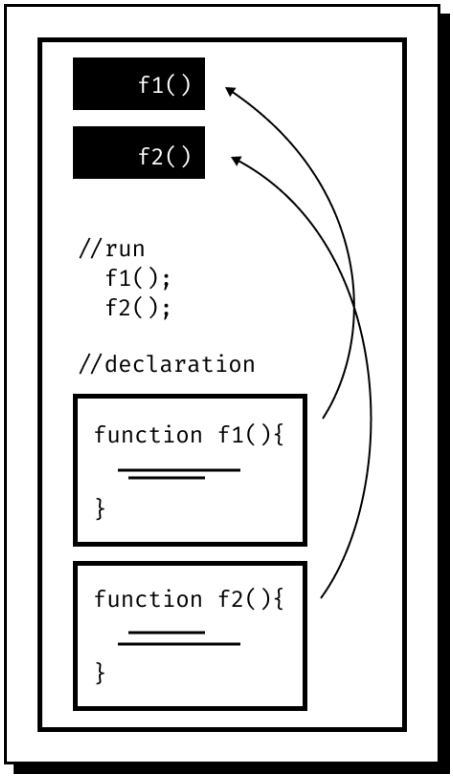
Sedangkan hal ini tidak berlaku untuk Function Expression.

```
let dinoName = 'brachio';

updateDinoName(dinoName); // ReferenceError: Cannot access 'updateDinoName'
before initialization

const updateDinoName = function (name) {
  name = 't-rex';
  return name;
};
```

Hal ini disebut dengan **Function Hoisting**, yaitu sebuah mekanisme dimana JavaScript Engine akan memindahkan semua function declaration ke bagian atas source code sebelum dieksekusi.



Anonymous Function

Anonymous Function adalah function yang tidak memiliki nama.

```
function() {
    console.log('Hi Brachio');
}
```

Penggunaan anonymous function biasa ditemukan pada Arrow function dan Immediately Invoked Function Expression (IIFE).

Arrow Function

Bisa dibilang arrow function adalah bentuk function yang paling simple.

Karena syntaxnya yang cukup sederhana, arrow function semakin sering digunakan ketimbang bentuk function yang lain.

Contoh:

```
const updateDinoName = (name) => {
  name = 't-rex';
  return name;
};

let dinoName = 'brachio';

updateDinoName(dinoName); // call function
```

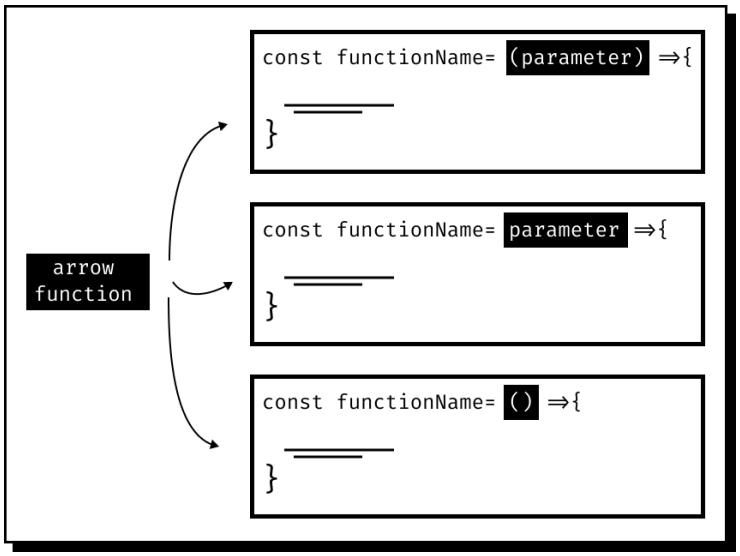
Kamu bisa menghilangkan **parentheses()** pada parameter sehingga menjadi:

```
const updateDinoName = name => {
  name = 't-rex';
  return name;
};
```

Contoh arrow function tanpa parameter:

```
const sayHi = () => console.log('Hi Brachio!');

sayHi(); // Hi Brachio
```



Contoh code pada pembahasan **array.map()** dapat lebih disederhanakan dengan mengubah bentuk function menjadi arrow function.

Dari sebelumnya:

```
const numbers = [1, 2, 3, 4, 5];

function doubleIt(number) {
  return number * 2;
}

const result = numbers.map(doubleIt);
```

Menjadi:

```
const numbers = [1, 2, 3, 4, 5];

const result = numbers.map((number) => number * 2);
```

Immediately Invoked Function Expression (IIFE)

IIFE adalah bentuk function yang dieksekusi segera setelah function dideklarasikan.

```
(function () {
  console.log('Hello Brachio');
})();
```

IIFE sering ditemui pada pemrograman JavaScript di browser terutama jika kita menggunakan plugin seperti jQuery.

Ketika sebuah function dideklarasikan, function tersebut akan ditambahkan ke window object.

```
function sayHi() {
  console.log('Hi Brachio');
}
// kita bisa memanggil function di atas dengan
window.sayHi;
```

Tidak hanya function, ketika kita mendeklarasikan sebuah variable, variable tersebut juga ditambahkan ke window object.

```
let word = 'Hi Brachio';
console.log(window.word); // 'Hi Brachio'
```

Hal ini tidak terlalu menjadi masalah jika nama function dan variable tersebut berbeda.

Tetapi seiring waktu baris code pada sebuah project JavaScript bisa berubah menjadi sangat panjang dan komplek, kita akan sulit mengingat setiap nama dari function dan variable yang sudah dideklarasikan.

Sehingga sangat mungkin terjadi kita memberi satu nama yang sama untuk dua function yang berbeda.

Oleh karena itu salah satu tujuan penggunaan IIFE adalah menghindari *Global NameSpace Pollution* yang dapat berakibat terjadinya *name collisions*, yaitu 'tabrakan' antara nama function satu dengan function yang lain atau bahkan nama function dengan nama variable.

Function Parameter

Multiple Parameter

Sebuah function dapat memiliki banyak parameter, namun seperti yang sudah dibahas pada Teknik Coding sebelumnya bahwa function sebaiknya memiliki parameter seminim mungkin.

```
const sendMessage = (dinoName, message) => {
  console.log(`Hi ${dinoName}, ${message}`);
};

const dinoName = 'Brachio';
const message = 'Where are you going?';
sendMessage(dinoName, message); // Hi Brachio, Where are you going?
```

Namun bagaimana jika ingin menggunakan banyak parameter ?

Salah satu cara yang bisa digunakan agar code tetap *clean* (mudah dibaca) adalah menggunakan *rest parameter*.

Default Parameter

Jika kita memiliki sebuah function yang memiliki dua parameter:

```
const sendMessage = (dinoName, message) => {
  console.log(`Hi ${dinoName}, ${message}`);
};
```

Kemudian kita hanya menyediakan salah satunya, maka salah satu valuenya adalah undefined.

```
const sendMessage = (dinoName, message) => {
  console.log(`Hi ${dinoName}, ${message}`);
};
const dinoName = 'Brachio';
sendMessage(dinoName); // Hi Brachio, undefined
```

Oleh karena itu kita dapat menambahkan Default Parameter untuk mengganti value undefined dengan value lain.

```
const sendMessage = (dinoName, message = 'How are you?') => {
  console.log(`Hi ${dinoName}, ${message}`);
};
const dinoName = 'Brachio';
sendMessage(dinoName); // Hi Brachio, How are you?
```

2.Konsep JavaScript sebagai Pendahuluan React

JavaScript Asynchronous

Single Threaded

Selain termasuk **Interpreted Language** dan **Dynamic-Typed Language**, JavaScript juga termasuk **Single Threaded Programming Language**.

Yaitu JavaScript hanya bisa melakukan satu operasi di satu waktu, sehingga code JavaScript dieksekusi secara berurutan dari atas ke bawah layaknya sebuah antrian atau biasa disebut synchronous.

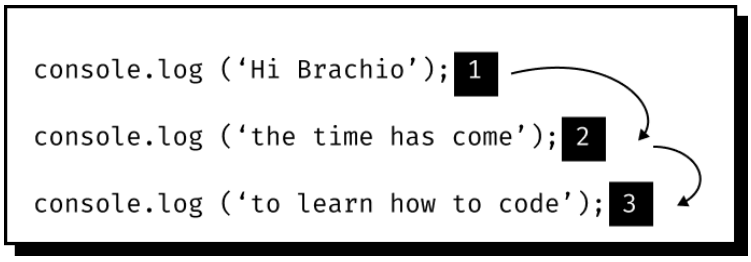
```
console.log('Hi Brachio');

console.log('the time has come');

console.log('to learn how to code');
```

Hasilnya:

```
Hi Brachio
the time has come
to learn how to code
```



Namun hal ini bisa menjadi masalah jika terdapat baris code yang eksekusinya membutuhkan waktu yang lama, seperti untuk mendownload data dari server.

Kita tidak tahu berapa lama waktu yang dibutuhkan untuk download data tersebut.

Jika mengikuti mekanisme synchronous dimana JavaScript hanya bisa melakukan satu operasi di satu waktu maka seharusnya JavaScript akan berhenti dan tidak akan mengeksekusi code selanjutnya sebelum download selesai.

Kenyataannya kita masih bisa melakukan aktifitas browsing meskipun download sedang berlangsung dan browser tidak *hang*.

Bagaimana JavaScript bisa melakukannya?

Jawabannya adalah dengan membuat JavaScript **asynchronous**, meskipun tidak benar-benar asynchronous.

Pada konsep asynchronous, code akan dieksekusi tanpa menunggu proses eksekusi code lain selesai sehingga seakan-akan dieksekusi secara bersamaan.

Kita bisa menggunakan simulasi berikut:

```
console.log('Hi Brachio');

setTimeout(function () {
  console.log('the time has come');
}, 3000);

console.log('to learn how to code');
```

setTimeout pada code di atas membuat kata *the time has come* akan ditampilkan setelah 3 detik.

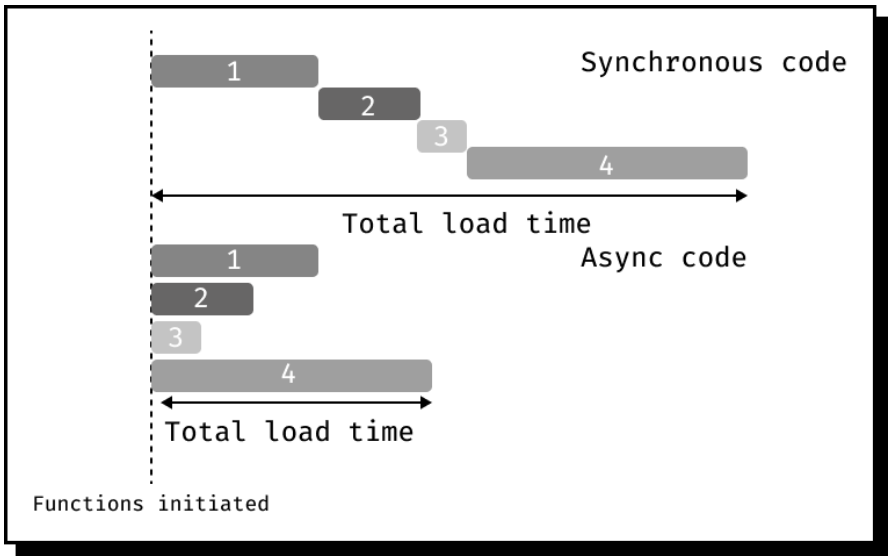
Namun ketika code diatas dieksekusi, JavaScript tidak akan menunggu selama 3 detik tapi akan segera menampilkan kata *to learn how to code*.

Sehingga hasilnya:

```
Hi Brachio
to learn how to code
the time has come
```

Function setTimeout() membuat JavaScript mengeksekusi code di atas secara asynchronous.

Berikut gambaran perbedaan antara synchronous dan asynchronous.



Event Loop

Untuk lebih memahami asynchronous pada JavaScript kita perlu mengetahui apa itu Event Loop.

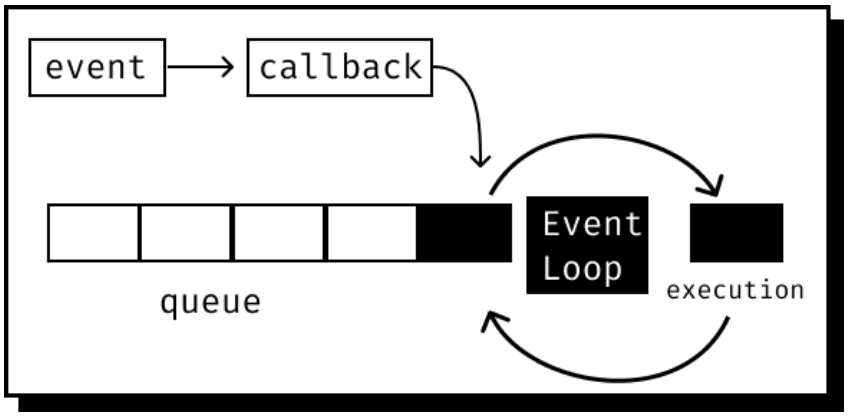
Event loop adalah bagian dari JavaScript Runtime yang bertugas untuk menangani Event Callback, Event Callback sendiri adalah bagian dari code yang dieksekusi setelah event tertentu.

Contoh Kasus: Klik tombol Download di browser.

- mouse click adalah **event**
- function yang dieksekusi setelah tombol di klik adalah **callback**

Ketika suatu event terjadi maka callback dari event tersebut akan ditempatkan pada suatu tempat yang disebut **Event Handler Queue** atau **Queue**.

Event Loop akan terus memonitor Queue dan akan mengeksekusi callback sesuai urutan siapa yang pertama masuk ke dalam Queue.



Teknik Asynchronous JavaScript

Ada 3 teknik yang digunakan untuk menghandle proses asynchronous pada JavaScript:

1. Callback
2. Promise
3. Async Await

Callback

Callback adalah function yang menjadi argument untuk function lain dan akan dieksekusi pada poin tertentu, bisa jadi saat ini atau nanti.

Contoh:

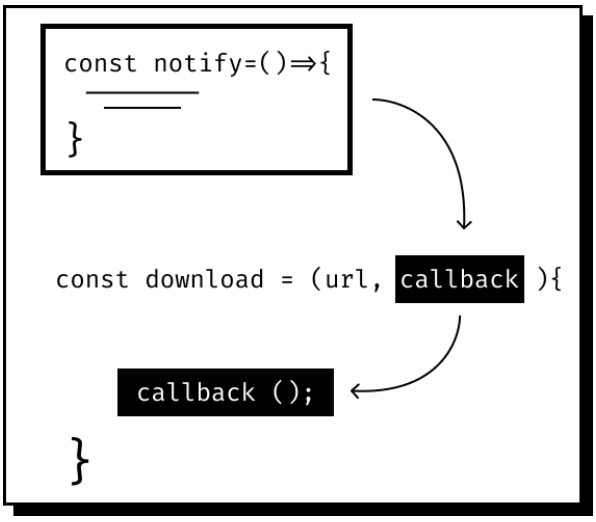
```
const notify = () => {
  console.log('Download complete!');
};

const download = (url, callback) => {
  console.log(`Downloading from ${url}....`);

  callback();
};

const url = 'https://brachio.site/download';

download(url, notify);
```



Output dari code di atas adalah:

```
Downloading from https://brachio.site/download....
Download complete!
```

Pada code di atas function notify adalah callback function, dipanggil setelah code **console.log(Downloading from \${url}....);**.

Menjadikan function sebagai argument untuk function yang lain adalah sesuatu yang mungkin dilakukan di JavaScript, karena function pada JavaScript adalah **First-Class Object**.

Yang berarti function memiliki karakter yang sama dengan object.

Sehingga sebuah function dapat:

- disimpan dalam sebuah variable, object atau array
- menjadi argument untuk function lain (High-Order Function)
- menghasilkan function baru

Nested Callback

Kita bisa menambahkan callback di dalam callback.

```
const download = (url, callback) => {
  console.log(`Downloading from ${url}....`);

  callback();
};

const url1 = 'https://brachio.site/download';
const url2 = 'https://trex.site/download';
const url3 = 'https://stego.site/download';

download(url1, function () {
  download(url2, function () {
    download(url3, function () {
      console.log('Download complete!');
    });
  });
});
```

Code di atas dieksekusi tanpa error dan menghasilkan output yang sesuai dengan perkiraan.

```
Downloading from https://brachio.site/download....
Downloading from https://trex.site/download....
Downloading from https://stego.site/download....
Download complete!
```

Namun kondisi ini bisa mengarah ke suatu masalah yang disebut dengan **Callback Hell,** yaitu kondisi dimana terlalu banyak callback di dalam callback yang lain.

Callback Hell membuat code sulit untuk dipahami dan sulit diperbaiki jika ditemukan bug di dalamnya.

Sebagai gantinya kita bisa menggunakan **Promise**.

Promise

Promise bisa dikatakan sebagai object yang menyimpan hasil dari sebuah operasi asynchronous baik itu hasil yang diinginkan (*resolved value*) atau alasan kenapa operasi itu gagal (*failure reason*).

Kita ambil contoh seperti saat memesan ojek online.

Saat kita mencari driver lewat aplikasi, aplikasi akan berjanji(*promise*) memberi tahu hasil dari pencarian yang kita lakukan.

Hasil yang akan didapat adalah di antara dua kondisi, yaitu driver ditemukan (*resolved value*) atau alasan kenapa driver tidak ditemukan(*failure reason*).

Sebuah Promise berada di salah satu diantara 3 kondisi(state):

pending, operasi sedang berlangsung
fulfilled, operasi selesai dan berhasil
rejected, operasi selesai namun gagal

Sama seperti pada kasus memesan ojek online, status permintaan kita pada aplikasi online juga berada diantara tiga kondisi:

Mencari driver (**pending**)
Menemukan driver (**fulfilled**)
Driver tidak ditemukan (**rejected**)

Membuat Promise

Keyword yang dipakai untuk membuat Promise adalah **Promise**.

```
let progress = 100;

const download = new Promise((resolve, reject) => {
  if (progress === 100) {
    resolve('Download complete');
  } else {
    reject('Download failed');
  }
});
```

Function setelah keyword **new Promise** disebut **executor**.

Dan di dalam executor terdapat dua callback function:

- **resolve(value)** adalah callback function yang dieksekusi jika operasi yang dijalankan oleh executor berhasil(fulfilled)
- **reject(error)** adalah callback function yang akan dieksekusi jika operasi gagal (rejected)

Promise Handler

Selanjutnya untuk merespon hasilnya (baik berupa value atau error) kamu perlu menambahkan **handler**.

Handler biasanya berupa function dan ditempatkan di dalam method bernama **then()**.

```
download.then((result, error) => {
  console.log(result); // Download complete
  console.log(error); // Download failed atau undefined jika tidak ada
error
});
```

sedangkan untuk merespon error kamu bisa tambahkan method **catch()**.

```
download
  .then((result) => {
    console.log(result); // Download complete
  })
  .catch((error) => {
```

```
    console.log(error); // Download failed atau tidak ditampilkan jika
tidak ada error
  });
```

Promise Chaining

Karena output dari method **then()** dan **catch()** adalah sebuah Promise, maka kamu bisa merangkainya(chain) dengan Promise yang lain.

```
let initProgress = 0;

const download = new Promise((resolve, reject) => {
  let progress = initProgress + 25;
  resolve(progress);
});

download
  .then((result) => {
    console.log(`Download progress is ${result}%`);
    return result + 25;
  })
  .then((result) => {
    console.log(`Download progress is ${result}%`);
    return result + 50;
  })
  .then((result) => {
    if (result === 100) {
      console.log('Download complete');
    }
  });
```

Promise.all()

Method **Promise.all()** digunakan untuk mengeksekusi Promise secara paralel.

Output dari Promise.all() adalah sebuah promise.

```
const downloadStart = new Promise((resolve, reject) => {
  resolve('0%');
});
```

```
const downloadHalf = new Promise((resolve, reject) => {
  resolve('50%');
});
const downloadFull = new Promise((resolve, reject) => {
  resolve('100%');
});

Promise.all([downloadStart, downloadHalf, downloadFull]).then((result) => {
  console.log(result); // [ '0%', '50%', '100%' ]
});
```

Callback vs Promise

Seperti yang disebutkan sebelumnya bahwa kita bisa menggunakan promise untuk mengatasi masalah *callback hell*.

❌ Callback Hell

```
const download = (url, callback) => {
  console.log(`Downloading from ${url}....`);

  callback();
};

const url1 = 'https://brachio.site/download';
const url2 = 'https://trex.site/download';
const url3 = 'https://stego.site/download';

download(url1, function () {
  download(url2, function () {
    download(url3, function () {
      console.log('Download complete!');
    });
  });
});
```

✔️ Promise

```
const download = (url) => {
  return new Promise((resolve, reject) => {
    resolve(`Downloading from ${url}....`);
  });
};

const url1 = 'https://brachio.site/download';
const url2 = 'https://trex.site/download';
const url3 = 'https://stego.site/download';

Promise.all([download(url1), download(url2), download(url3)]).then((result)
=> {
  for (let downloadInfo of result) {
    console.log(downloadInfo);
  }
  console.log('Download Complete');
});
```

Callback adalah function dan Promise adalah object

Async Await

Async/Await diperkenalkan di ES8 / ES2017 untuk menghandle operasi asynchronous dengan syntax yang lebih mirip dengan synchronous.

Async/Await sendiri dibuat di atas Promise.

Kita bisa membuat simulasi proses download menggunakan code dibawah ini.

```
const getStatus = (url) => {
  console.log(`Downloading from ${url}...`);

  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve('Download Complete');
    }, 3000);
  });
};
```

```
async function download(url) {
  let status = await getStatus(url); // tunggu sampai promise selesai
  console.log(status);
}

const url = 'https://brachio.site/download';

download(url);
```

Pada code di atas keyword await membuat function download harus menunggu sampai function **getStatus()** selesai dieksekusi.

Keyword await harus berada di dalam function async

Promise vs Async Await

Sekarang mari kita update code sebelumnya yang menggunakan Promise menjadi menggunakan Async/Await.

Promise

```
const download = (url) => {
  return new Promise((resolve, reject) => {
    resolve(`Downloading from ${url}....`);
  });
};

const url1 = 'https://brachio.site/download';
const url2 = 'https://trex.site/download';
const url3 = 'https://stego.site/download';

Promise.all([download(url1), download(url2), download(url3)]).then((result)
=> {
  for (let downloadInfo of result) {
    console.log(downloadInfo);
  }
  console.log('Download Complete');
});
```

Async/Await

```
const download = (url) => {
  return new Promise((resolve, reject) => {
    resolve(`Downloading from ${url}....`);
  });
};

const url1 = 'https://brachio.site/download';
const url2 = 'https://trex.site/download';
const url3 = 'https://stego.site/download';

async function runDownload() {
  let result1 = await download(url1);
  console.log(result1);

  let result2 = await download(url2);
  console.log(result2);

  let result3 = await download(url3);
  console.log(result3);

  console.log('Download Complete');
}

// run async function
runDownload();
```

JavaScript Module

Semakin berkembang suatu aplikasi maka semakin banyak pula baris code yang ditulis, pada satu file JavaScript mungkin terdapat lebih dari 1000 baris code.

Agar lebih mudah dikelola, kamu bisa memecah file JavaScript tersebut menjadi beberapa file JavaScript yang disebut **module**.

Ada beberapa sistem module yang bisa diimplementasikan di JavaScript, seperti ES Module dan CommonJS module. Di bagian ini jenis module yang digunakan adalah ES Module.

Export & Import

Kamu bisa export function, variable, object atau semua yang ada dalam satu module untuk kemudian di import ke module yang lain.

Beberapa contoh bentuk export & import:

Export & Import variable dan function satu per satu.

Filename: **profile.js**

```
export const dinoName = 'brachio';

export const sayHi = (name, message) => {
  console.log(`Hi ${name}, ${message}`);
};
```

Filename: **main.js**

```
import { dinoName, sayHi } from './profile.js';

// run
sayHi(dinoName, 'How are you?');
```

Export & Import semua yang ada di dalam module sekaligus

```
const dinoName = 'brachio';

const sayHi = (name, message) => {
  console.log(`Hi ${name}, ${message}`);
};

export {dinoName, sayHi};
```

```
import * as profile from './profile.js';

console.log(profile.dinoName);
profile.sayHi(profile.dinoName, 'How are you?');
```

Export & Import semua yang ada di dalam module sekaligus dengan nama atau alias

```
const dinoName = 'brachio';

const sayHi = (name, message) => {
  console.log(`Hi ${name}, ${message}`);
};

export {dinoName as name, sayHi};
```

```
import {name, sayHi as speak} from './profile.js';

// run
speak(name, 'How are you?');
```

Default Export & Import

Hanya satu default export per module.

```
export const dinoName = 'brachio';

const sayHi = (name, message) => {
  console.log(`Hi ${name}, ${message}`);
};
```

```
export default sayHi;
```

Dengan import default, **curly braces {}** dapat dihilangkan saat import dan menggunakan nama yang berbeda.

```
import speak from './profile.js';

// run
speak('brachio', 'How are you?');
```

JavaScript Class

Pada bahasa pemrograman seperti Java atau C# konsep **class** sangat familiar bahkan penggunaan class cukup mendominasi dalam pembuatan aplikasi Java atau C#.

Class bisa didefinisikan sebagai kumpulan variable dan method yang dipakai sebagai blueprint atau rancangan untuk membuat sebuah object(bukan JavaScript Object).

Apakah JavaScript juga memiliki konsep class?

Asalnya JavaScript tidak memiliki konsep class, karena JavaScript bukan **Object Oriented Programming(OOP)** language tapi **Prototype-based language**.

Namun pada tahun 2015, ES6 memperkenalkan konsep class, yang mirip dengan class pada OOP language meskipun tidak benar-benar mirip.

Syntax class pada JavaScript:

```
class className {
    constructor()

    method1()
    method2()
    ...
}
```

Contoh:

```
class Dino {
  constructor(name) {
    this.name = name;
  }

  weight = 100;

  sayHi() {
    console.log(`Hi my name is ${this.name}`);
  }
}
```

```
const blueDino = new Dino('brachio');
const redDino = new Dino('t-rex');

//call

console.log(blueDino.weight); //100
console.log(redDino.weight); //100

blueDino.sayHi(); // Hi my name is brachio
redDino.sayHi(); // Hi my name is t-rex
```

Pada code di atas kita membuat dua object/instance baru yaitu **blueDino** & **redDino** yang keduanya memiliki 'karakter' yang sama karena sama-sama menerapkan *blueprint* dari class Dino.

Instance adalah sebuah object perwujudan dari class, memiliki variable dan method yang sama dan sesuai dengan apa yang digambarkan oleh class.

Bagian-bagian Class

1. Constructor
Special method yang dieksekusi sebelum method yang lain.

2. Field
Variable yang dideklarasikan di dalam class.

3. Method
function yang dideklarasikan di dalam class.

4. Getter/Setter
method yang digunakan untuk mengakses value dan mengubah value di dalam class

Contoh:

```
class Dino {
  //constructor
  constructor(name, age, weight) {
    this.name = name;
    this.age = age;
    this.weight = weight;
  }
```

```
  // getter
  get name() {
    return this.name;
  }




  // setter
  set name(newName) {
    this.name = newName
  }

  // method
  greeting() {
    console.log(`Hi my name is ${this.name}`);
  }
}
```

Destructuring

Destructuring adalah javascript *expression* yang digunakan untuk menyimpan value dari suatu array atau object ke dalam dalam variable yang berbeda.

Jika ingin menyimpan properties dari sebuah object ke dalam variable maka kamu bisa melakukannya seperti ini:

```
const blueDino = {
    firstName: 'brachio',
    lastName: 'saurus',
    address: 'Jakarta'
};

const firstName = blueDino.firstName;
const address = blueDino.address;

console.log(firstName);
console.log(address);
```

Dengan destructuring kamu bisa mendapat hasil yang sama:

```
const blueDino = {
    firstName: 'brachio',
    lastName: 'saurus',
    address: 'Jakarta'
};

const {firstName, address} = blueDino; // object destructuring

console.log(firstName);
console.log(address);
```

Array Destructuring

Destructuring juga bisa digunakan untuk array.

```
const dinoGroup = ['t-rex', 'brachio', 'tricera'];
```

```
const [redDino, blueDino] = dinoGroup;

console.log(redDino);
console.log(blueDino);
```

Spread Syntax & Rest Parameter

Keduanya sama-sama menggunakan **three dots(...)**.

Lalu apa bedanya dan apa fungsinya?

Spread Syntax (...)

Spread syntax digunakan untuk memisahkan semua item pada iterable object seperti array menjadi item tersendiri.

Penggunaan Spread Syntax

Clone Array atau Object

```
const dinoGroup = ['t-rex', 'brachio', 'tricera'];

const newDinoGroup = [...dinoGroup];
```

Merge Array atau Object

```
const dinoGroup1 = ['t-rex', 'brachio', 'tricera'];
const dinoGroup2 = ['allo', 'stego'];

const newDinoGroup = [...dinoGroup1, ...dinoGroup2];
```

Menambahkan item baru ke array

```
const dinoGroup1 = ['t-rex', 'brachio', 'tricera'];
const dinoGroup2 = ['stego', ...dinoGroup1];
```

Rest Parameter (...)

Rest Parameter adalah kebalikan dari Spread Syntax, Rest Parameter adalah parameter yang akan mengumpulkan item-item yang terpisah menjadi satu.

Sebagai contoh, function ini adalah function yang akan menjumlahkan semua argumen yang diterima.

```
const add = (numA, numB, numC) => {
  return numA + numB + numC;
};

const result = add(1, 2, 3);

console.log(result); // 6
```

Kamu bisa menambahkan rest parameter pada function di atas, sehingga menjadi:

```
const add = (...nums) => {
  let total = 0;

  for (let num of nums) {
    total += num;
  }

  return total;
};

add(1, 2, 3);

const result = add(1, 2, 3);

console.log(result); // 6
```

Karena semua argumen akan dijadikan ke dalam satu array maka kamu bisa menambah argument sebanyak mungkin.

```
add(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
```

Yang perlu diperhatikan adalah rest parameter harus menjadi parameter yang terakhir.

Jika kamu menaruh rest parameter selain di bagian akhir maka kamu akan mendapatkan error.

```
const add = (numA, ...nums, numB) => { // error
    ...
}
```

3. Teknik Coding JavaScript

Teknik coding yang akan dibahas disini adalah Teknik Shorthand dan JavaScript Style Guide.

Teknik Shorthand adalah teknik menulis dalam versi lain yang lebih ringkas.

Shorthand

Beberapa teknik Shorthand pada JavaScript yang bisa digunakan:

1. Ternary Operator

- ❏ Longhand (versi Panjang)

```
let color = 'red';
let dinoName;

if (color === 'red') {
  dinoName = 't-rex';
} else {
  dinoName = 'brachio';
}
```

- ❏ Shorthand

```
let color = 'red';

const dinoName = color === 'red' ? 't-rex' : 'brachio';
```

2. Short-circuit Evaluation

Short-circuit Evaluation sering digunakan untuk memvalidasi sebuah value sebelum disimpan ke dalam variable.

- ❏ Longhand

```
let redDino = 't-rex';
let blackDino;

if (redDino !== null || redDino !== undefined || redDino !== '') {
  blackDino = redDino;
} else {
  blackDino = 'allo';
}
```

- ❏ Shorthand

```
let redDino = 't-rex';

const blackDino = redDino || 'allo';
```

3. If

- ❏ Longhand

```
if(isDino === true)
```

- ❏ Shorthand

```
if(isDino)
```

4. Object Property

- ❏ Longhand

```
const redDino = 't-rex';
const blueDino = 'brachio';

const dino = {
  redDino: redDino,
  blueDino: blueDino
};
```

- ❏ Shorthand

```
const redDino = 't-rex';
const blueDino = 'brachio';

const dino = {
  redDino,
  blueDino
};
```

5. Object Values

- ❏ Longhand

```
const dino = {
  color: 'red',
  height: '1.5 m',
  weight: 100
};

const array = [ dino.color, dino.height, dino.weight ];

// output [ 'red', '1.5 m', 100 ]
```

- ❏ Shorthand

```
const dino = {
  color: 'red',
  height: '1.5 m',
  weight: 100
};

const array = Object.values(dino);

// output [ 'red', '1.5 m', 100 ]
```

6. Convert string to Number

- ❏ Longhand

```
const dinoWeight = Number('100');
const dinoHeight = parseFloat('1.5');
```

- ❏ Shorthand

```
const dinoWeight = +'100';
const dinoHeight = +'1.5';
```

7. Destructuring

Lihat pada pembahasan Destructuring

8. AND (&&) Operator

- ❏ Longhand

```
const dino = true;

const sayHi = () => console.log('Hi');

if(dino) {
  sayHi();
}
```

- ❏ Shorthand

```
const dino = true;

const sayHi = () => console.log('Hi');

dino && sayHi();
```

9. Null, Undefined, Empty Checks

- ❏ Longhand

```
if (data !== null || data !== undefined || data !== '') {
    let dino = data;
}
```

- ❏ Shorthand

```
let dino = data || '';
```

10. Nullish coalescing Operator

Return value sebelah kanan jika value kiri adalah **null** atau **undefined**

```
const dino = null ?? 'brachio';
console.log(dino); // brachio
```

11. Foreach Loop

- ❏ Longhand

```
const data = [1,2,3,4,5];

for (let i = 0; i < data.length; i++) {
  console.log(data[i]);
}
```

- ❏ Shorthand

```
const data = [1,2,3,4,5];

for (const num of data) console.log(num);
```

12. Switch

❏ Longhand

```
const number = 1;
const printBlue = () => console.log('blue');
const printRed = () => console.log('red');

switch (number) {
  case 1:
    printBlue();
  break;
  case 2:
    printRed();
  break;
}
```

❏ Shorthand

```
const number = 1;
const printBlue = () => console.log('blue');
const printRed = () => console.log('red');

const prints = {
  1: printBlue,
  2: printRed,
}

prints[number] && prints[number]();
```

JavaScript Style Guide

Dalam menulis code di JavaScript banyak sekali pilihan atau bentuk yang bisa digunakan.

Contohnya:

- Dalam membuat string kamu bisa memilih antara menggunakan **double quote** atau **single quote**
- Bebas memilih antara menambahkan **semicolon (;)** di akhir baris atau tidak menggunakannya sama sekali
- Dalam membuat function kita bisa menggunakan **function declaration**, **function expression** atau **arrow function**
- dll

Mungkin timbul pertanyaan, bagaimana code yang ditulis bisa konsisten jika ada terlalu banyak pilihan bentuk dimana antara satu bentuk dengan bentuk yang lain terkadang tidak memiliki perbedaan yang berarti?

Apalagi jika harus bekerja dalam sebuah team dimana setiap developer besar kemungkinan memiliki style atau cara menulis code masing-masing.

Untuk mengatasi masalah ini kamu bisa menggunakan JavaScript Style Guide / JavaScript Standard yaitu standard yang bisa digunakan dalam menulis code JavaScript.

Sehingga code menjadi lebih konsisten serta tidak perlu menghabiskan banyak waktu untuk memilih.

Beberapa JavaScript Style Guide yang sering dipakai:

1. Airbnb JavaScript Style Guide
2. Google JavaScript Style Guide
3. Standard JavaScript Style Guide

Kamu bisa memilih salah satu dari JavaScript Style Guide yang ada dan menggunakannya mulai dari awal hingga akhir pembuatan aplikasi.

Linter

Selain menerapkan JavaScript Style Guide kamu mungkin juga membutuhkan **Linter**.

Linter adalah tool untuk menganalisa code untuk menemukan error, bug, typo atau code smell serta memberikan info bagaimana menghindari atau memperbaiki error yang ditemukan.

Salah satu linter yang umum digunakan adalah ESLint.

Kamu juga bisa menggabungkan antara ESLint dengan JavaScript Style Guide, sehingga ESLint akan menganalisa code sesuai standar dari JavaScript Style Guide yang digunakan.

Untuk pembahasan yang lebih lengkap silahkan lihat pada pembahasan **Tool -> Linter**.

4. Fitur-fitur baru JavaScript

Hampir setiap tahun muncul versi terbaru dari ECMAScript. Pada setiap versinya diperkenalkan berbagai fitur baru.

Berikut beberapa fitur terbaru JavaScript:

ES2020/ES11

1. **BigInt**

Tipe data untuk menyimpan angka lebih besar dari **9007199254740992**

```
const bigInt = BigInt(9007199254740991) // 9007199254740991n
```

2. **Dynamic Import**

Import file JavaScript sebagai module

```
import('/modules/dino-module.js')
  .then((module) => {
    // Do something with the module.
  });
```

3. **Nullish Coalescing Operator (??)**

Return value sebelah kanan jika value kiri adalah **null** atau **undefined**

```
const dino = null ?? 'brachio';
console.log(dino); // brachio
```

4. Optional Chaining (?.)

Akses properties dari object tanpa harus melakukan validasi secara terpisah.

```
const dino = {
  name: 'Brachio',
  color: {
    name: 'Red'
  }
};

const dinoName = dino.color?.name;
console.log(dinoName); // expected output: undefined
```

5. Promise.allSettled()

Return sebuah promises berisi hasil akhir dari setiap promise terlepas dari kondisi fulfilled atau rejected.

```
const promise1 = Promise.resolve(3);
const promise2 = new Promise((resolve, reject) => setTimeout(reject, 100,
'foo'));
const promises = [promise1, promise2];

Promise.allSettled(promises).
  then((results) => results.forEach((result) =>
console.log(result.status)));
```

6. String matchAll()

Return semua hasil yang sesuai dengan kriteria Regular Expression (regex)

```
const regexp = /t(e)(st(\d?))/g;
const str = 'test1test2';

const array = [...str.matchAll(regexp)];

console.log(array[0]);
// expected output: Array ["test1", "e", "st1", "1"]
```

```
console.log(array[1]);
// expected output: Array ["test2", "e", "st2", "2"]
```

ES2021/ES12

7. replaceAll()

Ganti semua string sekaligus.

```
const words = 'The dino is lazy, the dino is happy, the dino is a dino';

console.log(words.replaceAll('dino', 'bird'));
// The bird is lazy, the bird is happy, the bird is a bird
```

8. Promise.any()

Kebalikan dari Promise.all(), return promise setelah salah satu promise *fulfilled* (operasi selesai dan berhasil)

```
const promise1 = Promise.reject(0);
const promise2 = new Promise((resolve) => setTimeout(resolve, 100,
'dino'));
const promise3 = new Promise((resolve) => setTimeout(resolve, 500,
't-rex'));

const promises = [promise1, promise2, promise3];

Promise.any(promises).then((value) => console.log(value)); //dino
```

9. Numeric Separator

```
const number = 1_000_000;

console.log(number); // 1000000
```

10. Logical Assignment Operators

Logical AND Assignment &&=

Code dibawah ini

```
if (a) {
    a = b;
}
```

Bisa ditulis menjadi

```
a &&= b;
```

Logical OR Assignment ||=

```
if (!a) {
    a = b;
}
```

Bisa ditulis menjadi

```
a ||= b
```

Jika a bernilai **false**, maka **a = b**

Logical Nullish Assignment ??=

```
if (a == null) {
    a = b;
}
```

Bisa ditulis menjadi

```
a ??= b;
```

Selengkapnya bisa lihat disini.

Dengan memanfaatkan fitur-fitur yang sudah disebutkan diatas kamu bisa menulis code JavaScript dengan lebih mudah dan efisien. Tetapi harus tetap diperhatikan tingkat *readability* dari code yang ditulis. Menggunakannya secara berlebihan juga bukan ide yang bagus.