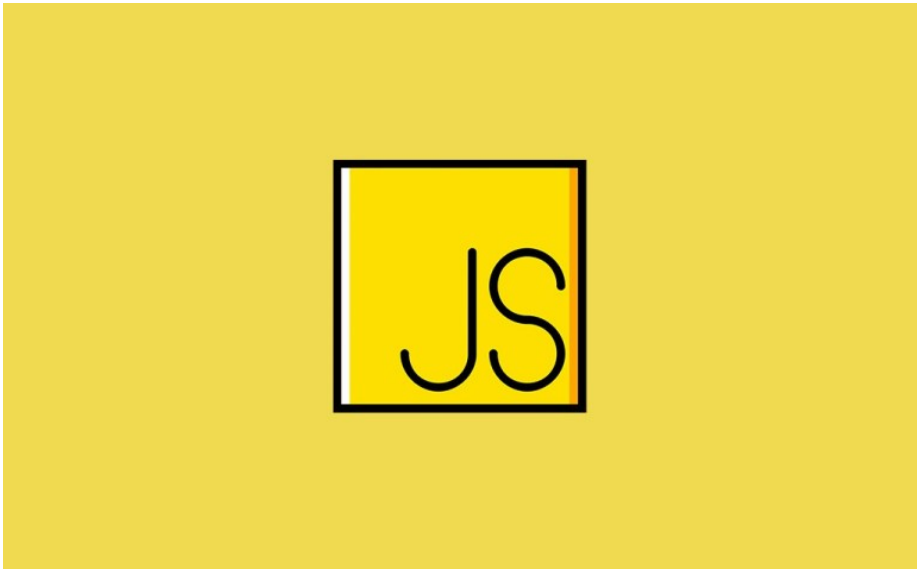
Konsep Penting di JavaScript Modern (ES6) yang Harus Diketahui



JavaScript merupakan bahasa pemrograman yang sedang tren pada tahun belakangan ini. Berbagai teknologi muncul menggunakan bahasa pemrograman JavaScript, mulai dari website, mobile, IOT, desktop.

Salah satu yang menarik perhatian adalah pada sisi website, di mana teknologi-teknolog baru muncul menggunakan bahasa pemrograman JavaScript. Pada sisi Frontend muncul Framework JavaScript seperti ReactJs, VueJs, dan Angular. Pada sisi Backend, muncul teknologi NodeJs dan Frameworknya yaitu ExpressJs.

Semuanya menggunakan bahasa pemrograman JavaScript, wow.

Namun, sebelum mempelajari berbagai Framework yang ada terutama pada sisi web baik Frontend atau Backend, terdapat konsep JavaScript dasar dan konsep JavaScript modern yang harus diketahui, dipelajari, dan dipraktikkan.

Pada tulisan kali ini, kita hanya fokus membahas beberapa konsep JavaScript modern. Tulisan ini akan terus berlanjut dan disempurnakan sesuai dengan perkembangan JavaScript.

Terakhir, singkatnya semakin baik kita memahami JavaScript (Native/Pure/Vanilla) maka semakin baik kita memahami Framework JavaScript.

Mari kita bahas konsep JavaScript modern yang ada.

Outline

- Arrow Functions
- Methods Array
- Destructing
- Default Parameters
- Spread Operator
- Short Conditionals

Variabel let dan const

Mengapa menggunakan let dan const?

- Keyword let dan const memberikan hasil yang lebih baik dan dapat diprediksi dibandingkan variabel yang dideklarasi dengan var.
- let dan const memperbaiki permasalahan yang terdapat pada keyword var seperti hoisting.
    - Hoisting merupakan fenomena di mana variabel bisa digunakan sebelum dibuat. Ini perilaku yang aneh dan bisa menyebabkan bug kecil di kode kita.
- let dan const bersifat block scope
    - Variabel yang dideklarasi dengan let dan const di dalam kurung kurawal (curly braces) hanya ada di dalam block tersebut dan tidak berada di luarnya.
    - Block scope menghasilkan kode yang mudah diprediksi dan meminimalisir error yang terjadi pada variabel.
- Keyword let digunakan untuk variabel yang bisa di-reassign setelah dideklarasi.
- Keyword const digunakan untuk variabel yang tidak bisa di-reassign.
    - variabel const bersifat konstan, artinya tidak mungkin berubah. Lebih tepatnya, variabel const tidak bisa di-reassign (ditugaskan kembali) atau diberikan nilai yang berbeda setelah dideklarasi.

Example

```
let greeting = "";
const newUser = true;

if (newUser) {
    // variabel let bisa di-reassign
    greeting = "Nice to meet you";
}
else {
    greeting = 'Welcome back';
}

// variabel const tidak bisa di-reassign
newUser = false; // hasilnya akan error

console.log(greeting);
```

Template Literals

Mengapa menggunakan Template Literals?

- Template literals sangat powerfull dibandingkan string biasa yang menggunakan kutip satu atau kutip dua.
- Interpolasi atau memasukkan nilai ke dalam string menjadi lebih mudah menggunakan sintaks `${}`.
- Tidak perlu menggunakan operator + untuk menggabungkan string.
- Lebih mudah menulis multiline
    - Tidak perlu menulis baris baru dengan karakter newline (\n) atau carriage return (\r)
- Bisa menggunakan tanda kutip bersarang (dengan kutip satu atau kutip dua) di dalam template literals.

Example

Menulis string menggunakan kutip dua.

```
/*
    menulis string menggunakan kutip dua.
    menggabungkan string menggunakan operator +
*/

const name = "Aufa Billah";
const greeting = "Hai" + name + ", Selamat Datang.";

console.log(greeting);
```

Menulis string menggunakan template literals.

```
/*
    menulis string menggunakan template literals
    memasukkan variabel dengan sintaks ${}
*/

const name = "Aufa Billah";
const greeting = `Hai ${name}, Selamat Datang`;

console.log(greeting);
```

Arrow Functions

Mengapa menggunakan Arrow Functions?

- Arrow function memungkinkan kita untuk menulis function dengan sintaks yang lebih singkat

        - Menggantikan keyword return dan body function dengan tanda panah: `=>`
    - Arrow function hadir dengan 3 shorthand, penulisan function lebih pendek:
        - Tanda kurung pada parameter bisa dihapus jika hanya ada 1 parameter.
        - Kurung kurawal (curly braces) pada body function dapat dihapus.
        - Tidak perlu keyword return karena arrow function memiliki implicit return (me-return secara default tanpa kurung kurawal).

Example

Penulisan fungsi normal.

```
// normal function
function capitalize(word) {
    return word.toUpperCase();
}

capitalize("Selamat Datang");
```

Penulisan arrow function.

```
// arrow function
const capitalize = (word) => {
    return word.toUpperCase();
}

capitalize("Selamat Datang");
```

Penulisan arrow function dengan shorthand.

```
// arrow function dengan shorthand
const capitalize = word => word.toUpperCase();

capitalize("Selamat Datang");
```

Methods Array

Mengapa menggunakan methods array?

- Dibandingkan menggunakan for untuk me-looping array, methods array seperti map, filter, reduce memungkinkan kita untuk me-looping array dengan tujuan tertentu:
    - map: mengubah setiap elemen array.

        - filter: memfilter item dari array yang tidak memenuhi kondisi yang diberikan.
        - reduce: mengubah seluruh array dengan cara apapun yang kita pilih (bahkan mengubah tipe data).
    - Method array lebih singkat dan lebih deklaratif dari pada for.
        - Ini bahkan lebih pendek ketika arrow function digunakan sebagai callback pada method array.

Example

Melakukan perulangan dengan for.

```
/*
    menyimpan name dari array users ke array usernames
    menggunakan for
*/

const users = [
    { name: "Bob", id: 1 },
    { name: "Jane", id: 2 },
    { name: "Fred", id: 3 },
];

const usernames = [];

for (let i = 0; i < users.length; i++) {
    usernames[i] = users[i].name;
}

console.log(usernames);
```

Melakukan perulangan dengan method array map.

```
/*
    menyimpan name dari array users ke array usernames
    menggunakan method array map
*/

const users = [
    { name: "Bob", id: 1 },
    { name: "Jane", id: 2 },
    { name: "Fred", id: 3 },
];

const usernames = users.map(user => user.name);

console.log(usernames);
```

Destructing

Mengapa menggunakan Destructing?

- Destructing memungkinkan kita untuk mengubah (mengekstrak) pasangan key-value object menjadi variabel.
    - Destructing membuat kita tidak perlu mengakses keseluruhan objek ketika ingin digunakan.
    - Dengan Destructing, kita bisa mengambil apapun nilai yang dibutuhkan dan disimpan ke variabel.
- Destructing juga bisa dilakukan pada array sama seperti object.

Example

Mengakses properti object tanpa destructing.

```
const user = {
    name: "Reed",
    username: "ReedBarger",
    email: "reed@gmail.com",
    details: {
        title: "Programmer"
    }
}

// mengakses object property tanpa destructing
console.log(`${user.name}, ${user.email}`);
```

Melakukan destructing pada object, lalu mengakses variabel hasil destructing.

```
const user = {
    name: "Reed",
    username: "ReedBarger",
    email: "reed@gmail.com",
    details: {
        title: "Programmer"
    }
}

// destructing object
const {name, email} = user;

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

Default Parameters

Mengapa menggunakan default parameters?

- Untuk menangani kondisi jika fungsi tidak diberikan argumen.

- Parameter default dapat mencegah error jika tidak ada nilai yang diberikan.

Example

Tanpa menggunakan default parameters.

```
// tanpa parameters default
function sayHi(name) {
    return `Hai ${name}`;
}

sayHi();
sayHi("Billah");
```

Menggunakan default parameters.

```
// menggunakan parameter default
function sayHi(name = "Aufa") {
    return `Hai ${name}`;
}

sayHi();
sayHi("Billah");
```

Arrow function dengan default parameters.

```
// Arrow function dengan parameter default
const sayHi = (name = "Aufa") => `Hi ${name}`;

sayHi();
sayHi("Billah");
```

Spread Operator

Mengapa menggunakan Spread Operator?

- Spread operator digunakan untuk menyebarkan object (pasangan key-value) ke object yang baru.
    - Spread operator hanya berjalan ketika membuat object atau array baru
- Spread operator bagus untuk membuat object baru dengan menggabungkan property mereka secara bersamaan.
    - Ketika sebuah object atau array disebarkan ke object atau array baru, salinan (shallow copy) akan dibuat untuk mencegah error.
- Spread operator dapat digunakan pada object dan array.

Example

```
const user = {
    name: "",
    email: "",
    phoneNumber: "",
};

const newUser = {
    name: "ReedBarger",
    email: "reed@gmail.com"
};

/*
    menggabungkan object user yang kosong dengan object newUser.
    object yang tersebar terakhir akan menimpa object sebelumnya jika propertinya memiliki nama yang sama.
*/

const mergedUser = {...user, ...newUser};

console.log(mergedUser);
```

Short Conditionals

Mengapa menggunakan Short Conditional?

- Ada cara yang lebih singkat untuk menulis kondisi if-else di JavaScript menggunakan ternary operator.
- Dibandingkan dengan statement if-else, ternary operator merupakan sebuah ekspresi.
- Lebih fleksibel menggunakan ternary di dalam ekspresi (seperti di dalam sintaks `${}` pada template literals).
- Ternary operator terkadang tidak disukai dibandingkan if-else, terutama jika ada lebih dari satu kondisi karena dalam kasus tersebut ternary operator sulit dibaca.

Example

Percabangan menggunakan if-else.

```
let age = 26;
let greeting;

// statement if-else
if (age > 18) {
    greeting = 'Hello, fellow adult';
}
else {
    greeting = 'Hey kiddo';
}

console.log(greeting);
```

Percabangan menggunakan ternary operator.

```
let age = 26;
let greeting;

// ternary operators
greeting = age > 18 ? "Hello, fellow adult" : "Hey
kiddo";

console.log(greeting);
```