**Glosarium**

**B**

**BOM**  
Singkatan dari Browser Object Model. Objek JavaScript yang tersedia di browser yang dapat digunakan untuk mengontrol browser.

**Browser**  
Aplikasi yang digunakan untuk mengakses sebuah website.

**C**

**CSS**  
Style sheet language yang digunakan untuk mendeskripsikan visual yang harus diterapkan pada markup language, salah satunya HTML.

**D**

**DOM**  
Singkatan dari Document Object Model, Objek JavaScript yang tersedia di browser yang dapat digunakan untuk mengontrol dan memanipulasi konten yang ditampilkan pada website.

**H**

**HTML**  
Markup language yang digunakan untuk menyusun struktur website.

**J**

**JavaScript**  
Bahasa pemrograman yang dapat digunakan oleh browser untuk membuat website tampak interaktif.

**L**

**Local Storage**  
Salah satu Browser API yang digunakan untuk menyimpan data tanpa ada batasan waktu. Data yang disimpan tidak akan hilang bila browser atau tabs browser ditutup kecuali jika kita menghapusnya.

**S**

**Server**  
Komputer yang dapat melayani permintaan dari komputer lainnya.

**Session Storage**  
Salah satu Browser API yang digunakan untuk menyimpan data sementara pada browser. Data akan hilang ketika browser atau tab browser ditutup.

## Daftar Referensi

[1] Dicoding, “Developer Academy: Belajar Dasar pemrograman web.” [Daring]. Tersedia: <https://www.dicoding.com/academies/123>. [Diakses: 29-Apr-2021].

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

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

## Pendahuluan Dasar JavaScript

Di kelas [Belajar Dasar Pemrograman Web](https://www.dicoding.com/academies/123), Anda sudah mampu membuat website sederhana menggunakan HTML dan CSS. Bila Anda paham dan menguasai kedua skill tersebut, sebenarnya Anda sudah mampu membuat website yang sangat amat cantik. Wah!

Kini timbul pertanyaan, apakah dengan HTML dan CSS saja sudah cukup dalam membuat website? Memang benar dengan HTML dan CSS saja Anda dapat membuat sebuah website, namun website hanya sebatas halaman informasi yang statis (seperti website yang Anda buat di kelas [Belajar Dasar Pemrograman Web](https://www.dicoding.com/academies/123)). Tidak akan ada interaktivitas dengan pengguna, kaku, dan sulit dalam menampilkan konten yang dinamis.

Lalu bagaimana agar kita dapat mengembangkan website yang interaktif dan dinamis? Perkenalkan, JavaScript!

JavaScript merupakan satu-satunya bahasa pemrograman yang dapat dipahami oleh browser untuk membantu website lebih interaktif dan dinamis. Dengan JavaScript, kita dapat memberikan fungsionalitas lebih pada website. Kita bisa membuat game di website, membuat website layaknya aplikasi, bahkan mengedit foto sekalipun. Luar biasa! JavaScript mampu mengubah fungsi dari website yang sebelumnya hanya sebatas portal untuk menampilkan informasi statis.

Sebelum kita melakukan hal-hal keren menggunakan JavaScript, tentu kita perlu mendalami dulu dasar dari bahasa pemrograman JavaScript. Kita akan belajar JavaScript untuk pengembangan web secara runut agar Anda tidak bingung.

Nah, di modul ini kita akan membahas secara detail dari dasar-dasar JavaScript hingga Anda mampu:

* Menuliskan sintaks JavaScript
* Mengetahui Statement dan Expressions
* Mengetahui cara menuliskan komentar di JavaScript
* Mengetahui cara membuat variabel
* Mengetahui tipe data yang ada di JavaScript
* Mengetahui berbagai macam operator yang ada di JavaScript
* Mengetahui cara melakukan percabangan di JavaScript
* Mengetahui cara melakukan perulangan di JavaScript
* Mengetahui cara mengelola data yang kompleks menggunakan objek dan array.
* Mengetahui mendeklarasikan atau membuat fungsi dan menggunakannya.

Supaya proses belajar menjadi lebih interaktif, apabila Anda memiliki pertanyaan perihal materi disajikan, jangan segan untuk bertanya via [Forum Diskusi](https://www.dicoding.com/academies/315/discussions). Mentor kami akan sigap untuk berusaha menjawab pertanyaan Anda. Selain itu, Anda juga dipersilahkan untuk membantu menjawab pertanyaan yang diajukan oleh peserta belajar lainnya.

## Apa itu JavaScript

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

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

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

1. console.log('Hello World');

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

## Sejarah JavaScript

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

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

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

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

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

## Mengenal Statement dan Expression dengan Menulis Kode JavaScript Pertama

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

1. console.log('Hello World');

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

[](https://www.dicoding.com/academies/315/tutorials/16640?from=16635)

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

console.log('Hello World!');

// Tampilkan nama Anda ke layar pada kode di bawah:

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

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

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

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

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

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

## Komentar

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

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

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

// Ini merupakan komentar satu baris, kode di bawah tidak akan dijalankan

// console.log('Halo!');

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

/\*

Ini merupakan komentar dengan lebih dari satu baris

Teks apa pun yang berada di sini akan dijadikan komentar.

Ketika menggunakan ini, jangan lupa untuk menutupnya.

\*/

## Variabel

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

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

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

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

1. let lastName;

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

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

let lastName;

lastName = 'Skywalker';

console.log(lastName);

/\*

Output:

Skywalker

\*/

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

let lastName = 'Skywalker';

console.log(lastName);

/\*

Output:

Skywalker

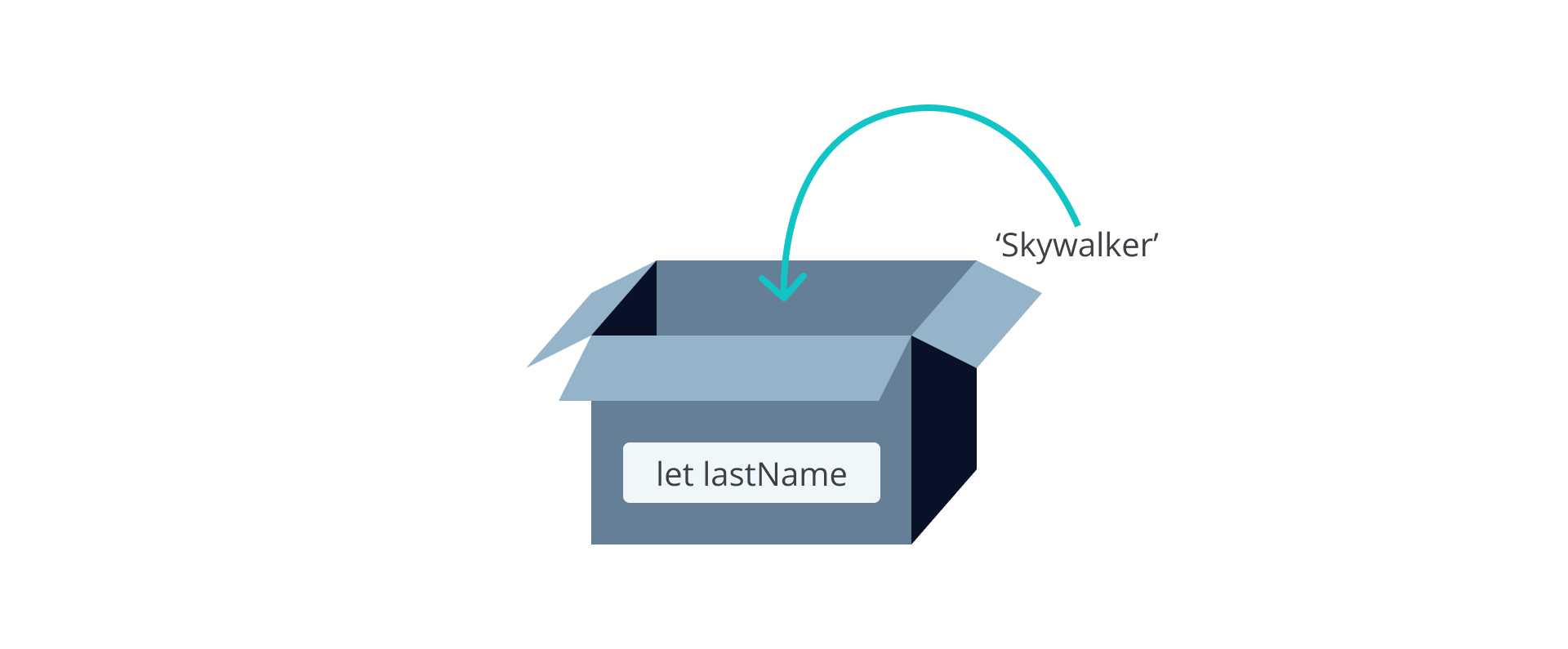
\*/

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

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

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

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

[](https://www.dicoding.com/academies/315/tutorials/16650?from=16645)

Variabel **lastName**di atas akan tersimpan di dalam memori komputer.

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

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

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

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

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

const z = 100;

console.log(z);

z = 200;

console.log(z);

/\* TypeError: Assignment to constant variable. \*/

## Tipe Data

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

### Undefined

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

1. let x;
2. console.log(typeof(x));
4. /\* Output: undefined \*/

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

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

### Numbers

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

1. let x = 10;

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

let x = 10;

console.log(typeof(x));

/\* Output: number \*/

let y = 17.25;

console.log(typeof(y));

/\* Output: number \*/

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

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

let a = 12;

let b = 9;

console.log(a + b);

console.log(a - b);

console.log(a \* b);

console.log(a / b);

console.log(a % b);

/\* Output:

21

3

108

1.3333333333333333

3

\*/

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

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

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

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

/\* Increment dan Decrement \*/

let postfix = 5;

console.log(postfix++); /\* Output: 5 \*/

console.log(postfix); /\* Output: 6 \*/

let prefix = 5;

console.log(++prefix); /\* Output: 6 \*/

### BigInt

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

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

const bigNumber = 1234567890123456789012345678901234567890n;

const myInt = 1234567890123456789012345678901234567890;

console.log(bigNumber);

console.log(myInt);

/\* Output:

1234567890123456789012345678901234567890n

1.2345678901234568e+39

\*/

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

1. const bigIntButSmall = 7n;

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

console.log(5n + 2n);

console.log(5n - 2n);

console.log(5n \* 2n);

console.log(5n / 2n);

console.log(5n % 2n);

/\* Output:

7n

3n

10n

2n (Bukan 2.5n)

1n

\*/

### Strings

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

let greet = 'Hello';

console.log(typeof(greet));

/\* Output: string \*/

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

const question = '"What do you think of JavaScript?" I asked';

console.log(question);

/\* Output: "What do you think of JavaScript?" I asked \*/

Lalu bagaimana

const answer = '"I think it's awesome!" he answered confidently'; // Invalid

console.log(answer);

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

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

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

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

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

let greet = 'Hello';

let moreGreet = greet + greet;

console.log(moreGreet);

/\* Output: HelloHello \*/

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

let greet = 'Hello';

let moreGreet = greet + " " + greet;

console.log(moreGreet);

/\* Output: Hello Hello \*/

let x = true;

let y = false;

console.log(typeof(x))

console.log(typeof(y))

/\* Output:

boolean

boolean

\*/

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

const a = 10;

const b = 12;

let isGreater = a > b;

let isLess = a < b;

console.log(isGreater);

console.log(isLess);

/\* Output:

false

true

\*/

### Null

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

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

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

1. let someLaterData = null;
2. console.log(someLaterData);
4. /\* Output:
5. null
6. \*/

### Symbol

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

const id = Symbol('id');

console.log(id);

/\* Output:

Symbol(id)

\*/

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

const id1 = Symbol('id');

const id2 = Symbol('id');

console.log(id1 == id2);

/\* Output:

false

\*/

Symbol ini umumnya digunakan sebagai nama properti dari Object. Object sendiri merupakan tipe data kompleks untuk menyimpan berbagai struktur data. Kita akan segera bertemu dan mempelajari tentang object nanti.

/\*\*

\* TODO:

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

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

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

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

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

\*/

// TODO

let firstName = "Aditya Imam";

let lastName = "Zuhdi";

let age = 21;

let isMarried = false;

console.log(firstName,lastName,age,isMarried);

/\*\*

\* Jangan hapus kode di bawah ini

\*/

module.exports = {

firstName, lastName, age, isMarried,

};

## Pengenalan Browser Object Model, Document Object Model, dan Event

Pada modul sebelumnya kita telah mempelajari seluk-beluk bahasa pemrogramanan JavaScript. Lantas, bagaimana JavaScript mempunyai pengaruh terhadap aspek Front-End dari sebuah halaman web? Dalam modul inilah kita akan menemukan keajaiban JavaScript untuk membuat halaman web kita menjadi interaktif.

Kita akan mempelajari hal baru yang bernama Browser Object Model (BOM), Document Object Model (DOM), dan Event. Kita akan berinteraksi dengan hal-hal tersebut melalui JavaScript sehingga kita bisa memperlakukan laman web dengan hal-hal unik yang belum tentu bisa dicapai dengan hanya menggunakan HTML dan CSS saja. Oleh karena itu, di modul ini diharapkan Anda dapat mengetahui dan memahami beberapa hal berikut.

* Browser Object Model (BOM)
* Member dari BOM beserta fungsinya
* Document Object Model (DOM)
* Bagaimana DOM Tree terbentuk
* Mampu mengubah konten website melalui teknik manipulasi DOM
* Event dan macam-macamnya
* Mampu menambahkan Event Handler pada HTML Element
* Custom Event serta mampu membuat dan menggunakannya
* Konsep dari Event Bubbling dan Event Capturing
* Event yang berada pada elemen <form>
* Event yang berada pada elemen <input>

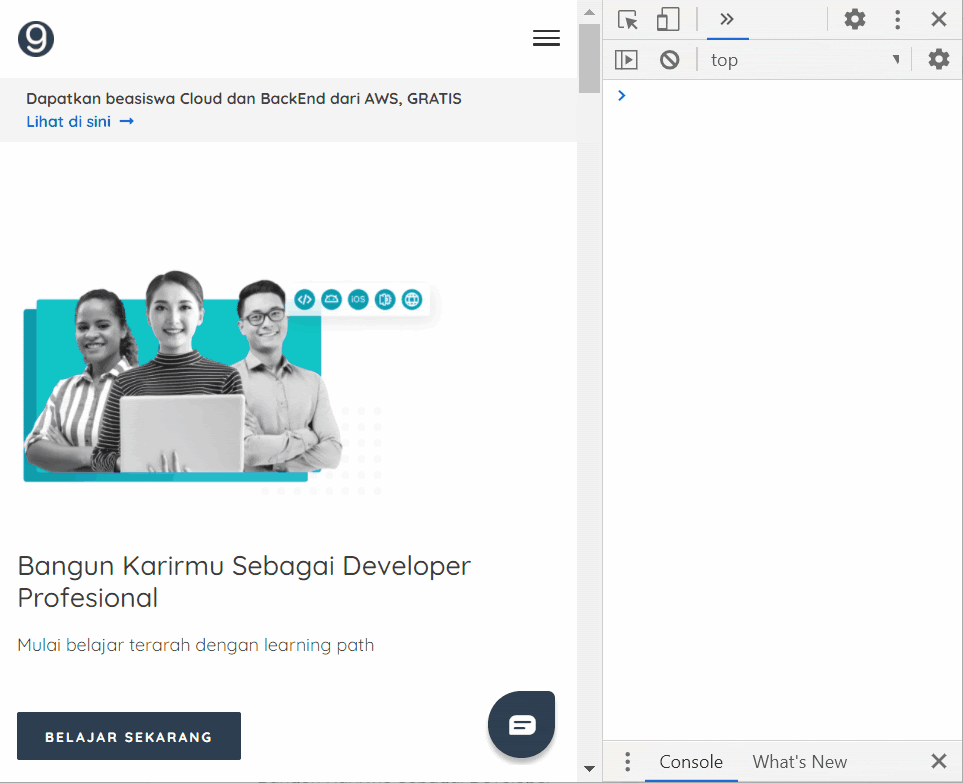
Supaya proses belajar menjadi lebih interaktif, apabila Anda memiliki pertanyaan perihal materi disajikan, jangan segan untuk bertanya via [Forum Diskusi](https://www.dicoding.com/academies/315/discussions). Mentor kami akan sigap untuk berusaha menjawab pertanyaan Anda. Selain itu, Anda juga dipersilakan untuk membantu menjawab pertanyaan yang diajukan oleh siswa lainnya.

Mari kita bersiap untuk membuat aspek Front-End Anda menjadi lebih keren. Let's Go!

## Cara JavaScript Mengontrol Website dan Browser

Setelah membaca materi sebelumnya, maka akan muncul pertanyaan “Bagaimana JavaScript mengontrol sebuah website atau browser?” Jawabannya ada dua, dan keduanya akan kita bahas di modul ini, yakni melalui **Browser Object Model (BOM)** dan **Document Object Model (DOM)**.

Yang pertama adalah **BOM**. Dengan **BOM**,kita dapat memberikan perintah-perintah khusus ke browser, misalnya melalui sebuah ‘atribut’ khusus milik browser yakni ‘window’ (akan kita membahasnya pada materi selanjutnya), sehingga kita bisa membuat browser menampilkan pesan pop-up. Caranya yakni menjalankan method alert() pada console milik browser. Berikut contohnya:



Selain method alert(), objek window juga memiliki method-method lainnya, seperti prompt, console, dsb. Tenang, kita akan berjelajah lebih jauh pada modul-modul selanjutnya.

Cara kedua adalah **DOM**. **DOM**sama seperti **BOM**. Perbedaannya adalah kita menggunakan global objek bernamadocument. Melalui global objek ini, kita bisa menangkap seluruh elemen dalam dokumen HTML guna memanipulasi konten HTML melalui method getElementById(). Method ini akan menangkap elemen berdasarkan value dari atribut id. Sebagai contoh, kita mengubah konten elemen HTML berikut ini.

## Menambahkan JavaScript di Website

Ketika membuat berkas html, tentu kita sudah tidak asing dengan tag html, bukan? Contohnya, seperti tag <html>, <style>, <p>, <div>, beserta tag-tag lainnya. Nah, jika kita sudah familier dengan gaya penulisan tag dalam sebuah berkas html, menambahkan kode JavaScript ke dalam berkas HTML akan lebih mudah. Mengapa demikian? Karena kita hanya perlu menggunakan tag <script> sebagai pembuka dan tag </script> sebagai penutup.

Ada dua cara untuk memasukkan kode JavaScript ke dalam berkas HTML kita, yakni secara **internal**dan **external**. Hmm, tampaknya terdengar familier? Betul! Hal ini mirip dengan cara menyematkan berkas CSS ke dokumen HTML, bukan?

Andaikan kita ingin meminta input dari user pada sebuah box yang muncul. Input tersebut berisi nama dari user dan akan kita munculkan kembali nama dari user tersebut pada sebuah box. Berikut adalah bentuknya dalam kode JavaScript:

1. const tamu = prompt('Siapakah Anda?');
2. alert('Selamat datang ' + tamu + '!');

Lalu kita siapkan juga berkas HTML bernama **index.html** dan strukturnyaseperti di bawah ini:

1. <!DOCTYPE html>
2. <html>
3. <body>
4. <h2 id="pesan">Selamat datang!</h2>
5. </body>
6. </html>

Nah, dari kedua kode program di atas, selanjutnya kita mempelajari cara memasukkan kode JavaScript ke dalam berkas HTML. Bagaimana caranya? Perhatikan pembahasan di bawah ini.

### Internal JavaScript

Pertama, kita akan membahas cara menulis sintaks untuk memasukkan kode JavaScript secara internal terlebih dahulu. Caranya cukup mudah layaknya menulis CSS secara internal pada berkas HTML. Cukup tuliskan kode JavaScript kita diantara tag pembuka dan penutup (<script> dan </script>). Contohnya adalah sebagai berikut.

1. <!DOCTYPE html>
2. <html>
3. <body>
4. <h2 id="pesan">Selamat datang!</h2>
6. **<script>**
7. **const tamu = prompt('Siapakah Anda?');**
8. **alert('Selamat datang ' + tamu + '!');**
9. **</script>**
10. </body>
11. </html>

Cukup sederhana, bukan?

### External JavaScript

Berikutnya kita akan membahas bagaimana memasukkan kode JavaScript ke dalam berkas HTML secara eksternal. Sama seperti memasukkan CSS secara eksternal pada berkas HTML, kita perlu menulis lokasi berkas berekstensi .js (dot js) pada atribut src di dalam tag <script> alias tag pembuka.

Sebagai contoh, kita akan memisahkan kode JavaScript sebelumnya pada berkas terpisah yang bernama **welcome.js**. Dengan demikian, untuk memasukkan **welcome.js** ke dalam berkas **index.html** dapat dituliskan seperti berikut.

1. <html>
2. <body>
3. <h2 id="pesan">Selamat datang!</h2>
4. **<script src="{direktori\_kamu}/welcome.js"></script>**
5. </body>
6. </html>

Pada tahap ini pasti kita akan bertanya-tanya, "Kenapa tidak menggunakan JavaScript internal saja? Padahal kan sama saja dan cara eksternal terkesan lebih kompleks!". Pertanyaan yang bagus!

Jawabannya sama dengan ketika kita ingin mengimplementasikan CSS secara internal dan eksternal. Kita menggunakan implementasi internal jika baris atau kompleksitas kode JavaScript sedikit atau rendah.

Namun, perlahan kita akan dibuat pusing jika jumlah kode JavaScript kita banyak serta memiliki kompleksitas yang tinggi. Salah satu kelebihan implementasi JavaScript secara eksternal adalah memudahkan kita untuk memperbaiki laman web ketika terdapat bugs atau error dan menambah konten ketika diperlukan. Kita pun dapat menggunakan berkas javascript pada multiple dokumen HTML sehingga kita tidak perlu menduplikasi kode. Tentu, hal ini sama seperti berkas CSS.

Contohnya pada halaman utama [W3C](https://www.w3.org/). Jika kita melihat berkas HTML-nya, terdapat sebuah elemen <script> yang merujuk ke sebuah berkas JavaScript eksternal.

Jika kita telaah kembali, di dalamnya terdapat kode JavaScript yang berisi perintah untuk memanipulasi struktur HTML mereka. Oleh karena itu, sebaiknya kita menggunakan cara JavaScript eksternal supaya dapat digunakan berulang kali.

### Dimana Harus Meletakkan tag <script>?

Walaupun implementasi JavaScript secara internal maupun eksternal terlihat mirip dengan CSS internal dan eksternal, tetapi terdapat perbedaan mendasar. Perbedaannya adalah kode CSS diletakkan dalam elemen <style> yang berada di dalam elemen <head>, sedangkan kode JavaScript diletakkan dalam elemen <script> yang terdapat di bagian akhir elemen <body>. Mengapa demikian?

Implementasi JavaScript ke dalam HTML akan berbeda jika dibandingkan dengan implementasi CSS. Hal tersebut karena tag <style> harus berada di dalam tag <head> yang membuat elemen-elemen pada dokumen HTML akan bergantung pada konten dalam CSS. Sehingga, langkah paling logis adalah mempersiapkan konten CSS terlebih dahulu.

Berbeda dengan JavaScript, jika website kita tidak memerlukan JavaScript untuk menampilkan halaman web, maka sebaiknya tag <script> diletakkan sebelum bagian akhir dari sebuah berkas HTML. Mengapa demikian? Karena jika meletakkannya di awal, semua konten HTML tidak akan dijalankan sebelum semua kode JavaScript yang berada pada tag <script> selesai dijalankan. Ambil contoh isi berkas HTML berikut:

1. <html>
2. <script>alert('Silakan menekan OK di bawah untuk memunculkan isi halaman ini');</script>
3. <body>
4. <h2 id="pesan">Selamat datang!</h2>
5. </body>
6. </html>

Jika dijalankan pada browser prosesnya akan seperti berikut:

Jika diperhatikan kembali pada tampilan di atas, kita harus menekan tombol "OK" terlebih dahulu supaya dapat menampilkan konten HTML. Hal ini dikarenakan elemen script dijalankan terlebih dahulu sehingga dokumen HTML belum di-render secara menyeluruh.

Anda harus benar-benar mempertimbangkan posisi peletakan tag <script> pada berkas HTML-nya. Jika tidak hati-hati, maka bisa saja tampilan halaman web menjadi tertunda dan dapat membuat pengalaman yang tidak mengenakkan bagi user yang sedang mengunjungi website kita.

Apakah ini berarti tag <script> harus selalu diletakkan di bagian akhir berkas HTML? Tidak juga. Jika tampilan halaman web Anda bergantung terhadap kode JavaScript, maka sebaiknya kamu letakkan di bagian yang sesuai.

## Browser Environment

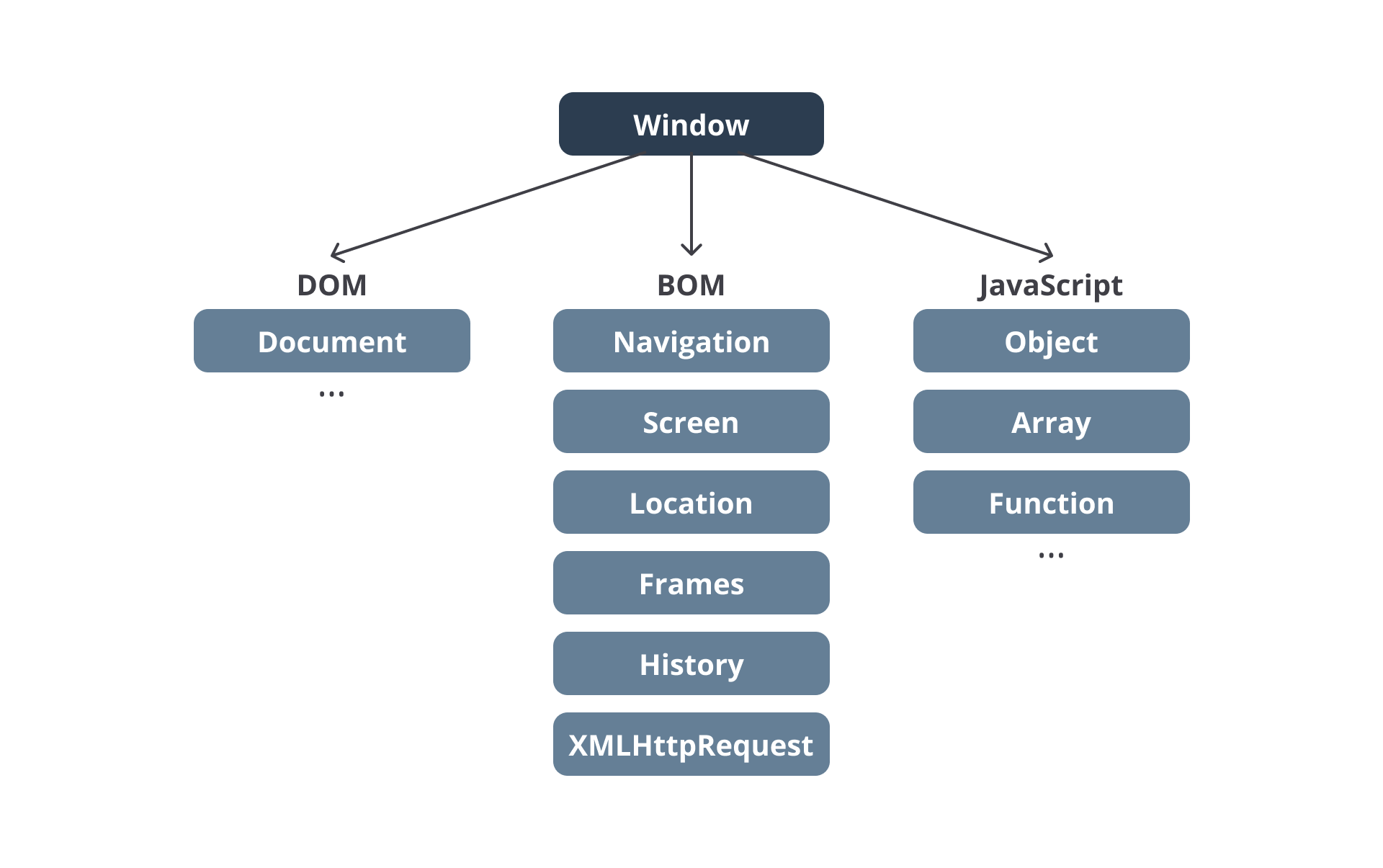
Selama belajar materi JavaScript yang dibahas di dalam kelas ini, apakah Anda menyadari semua kode JavaScript berjalan di perangkat browser dan tidak pada perangkat lainnya? Mengapa demikian? Karena kita menggunakan runtime environment milik browser alias semua kode JavaScript dijalankan di atas platform browser.

Pada awalnya Bahasa Pemrograman JavaScript didesain untuk berjalan di browser. Namun, seiring berjalannya waktu, kini banyak platform yang dapat menjalankan JavaScript di luar browser. Platform tersebut menggunakan runtime environment lain seperti Node JS. Salah satu contohnya adalah ketika kita menjalankan kode JavaScript pada [glot.io](https://glot.io/).

Walaupun JavaScript dapat berjalan di luar browser, kita perlu tahu bahwa JavaScript yang berjalan di browser memiliki **fungsionalitas khusus** yang tidak bisa ditemukan di tempat lain. Hal tersebut karena ia dijalankan di dalam **browser environment**.

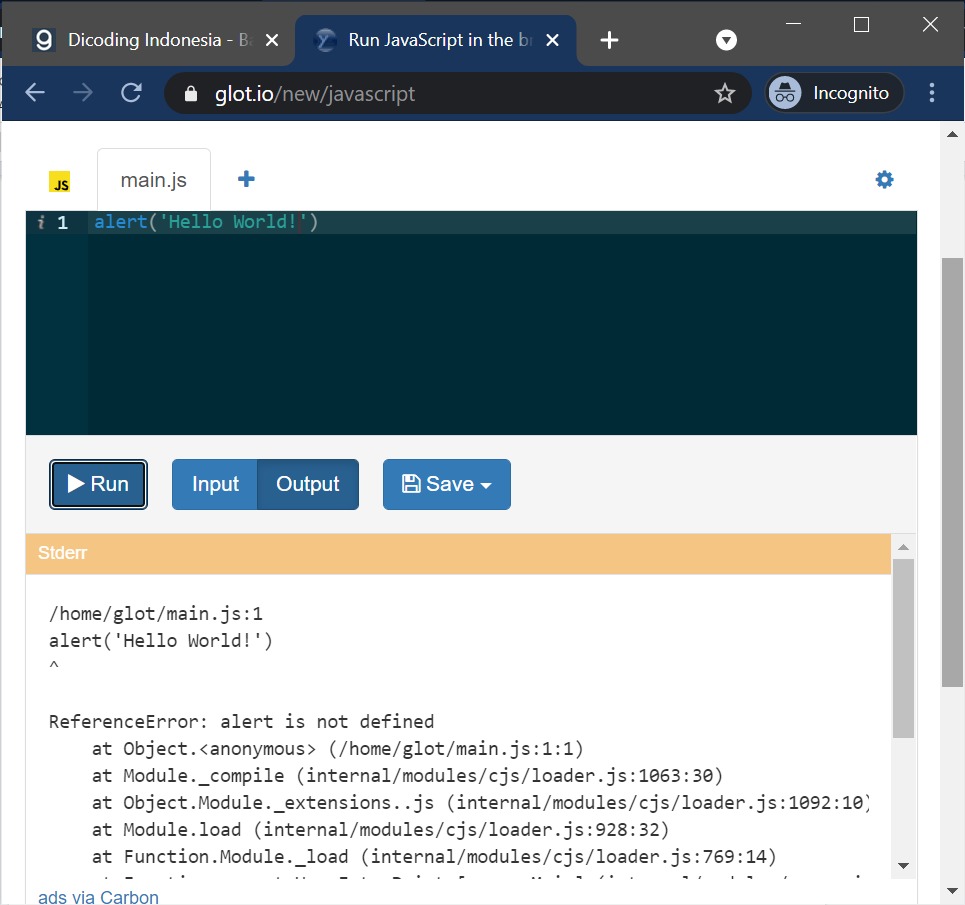
Apa istimewanya browser environment ini? Istimewanya terletak pada "peralatan-peralatan" khusus yang dapat digunakan oleh kode JavaScript untuk berinteraksi dengan browser maupun dengan dokumen HTML yang kita buat. Masih ingat dengan istilah **Browser Object Model** (BOM) dan **Document Object Model** (DOM)? Keduanya secara khusus hanya tersedia untuk JavaScript yang dijalankan dalam browser environment.

Pada gambar di bawah ini, kita melihat representasi dari objek window yang hanya bisa diakses oleh JavaScript dalam browser environment. Dengan melalui objek window, kita bisa mengakses DOM serta BOM.

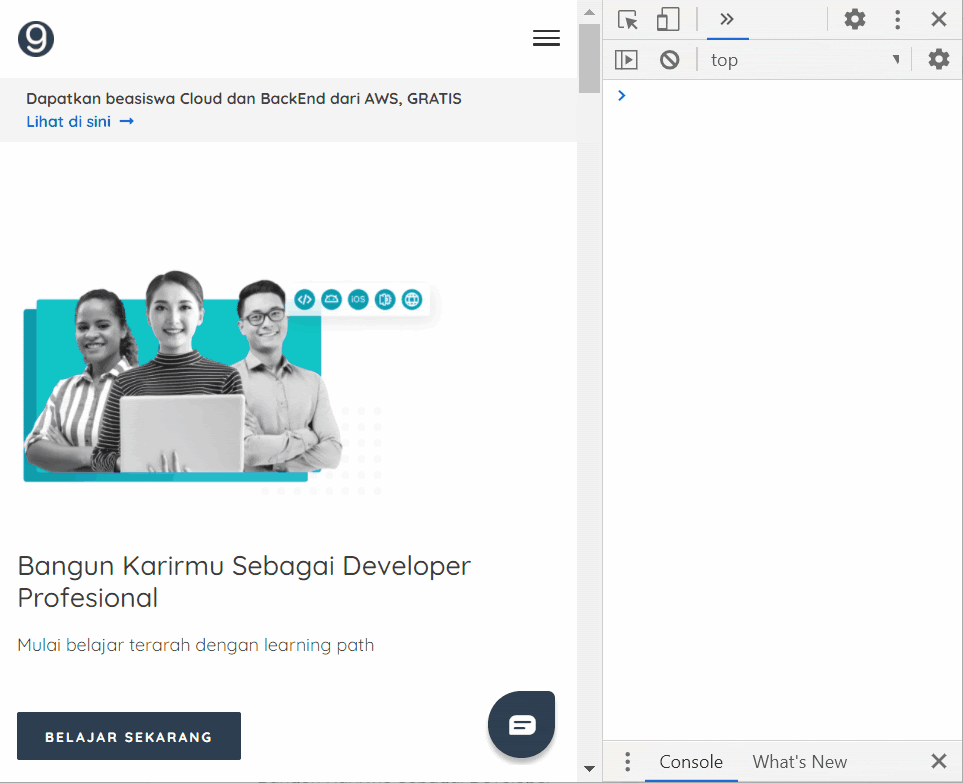
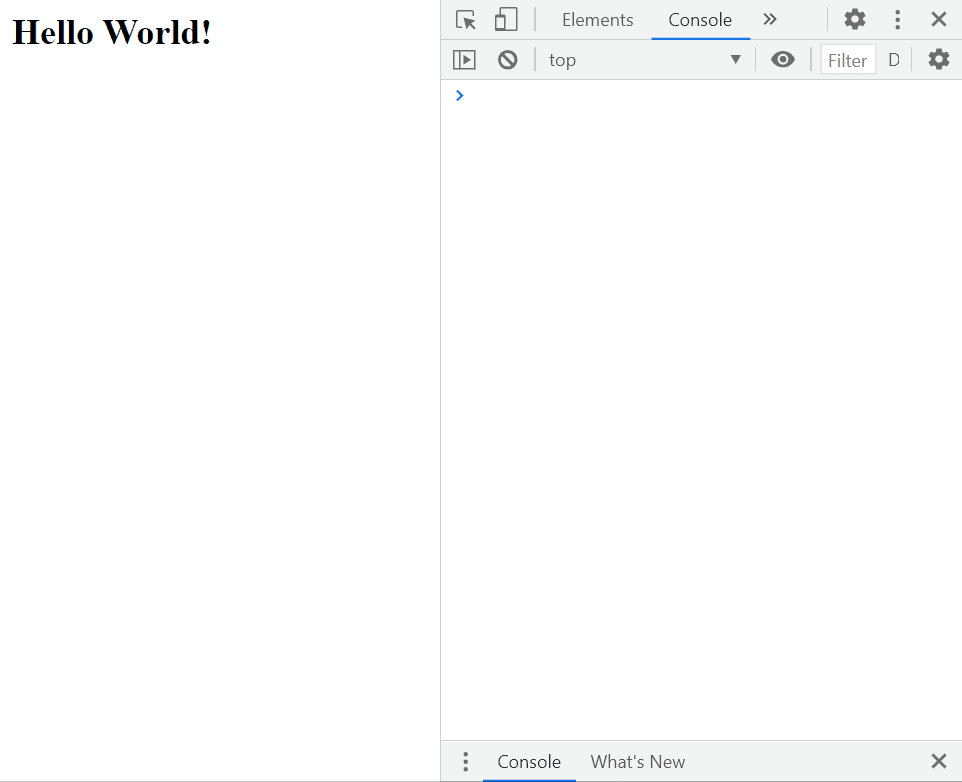
[](https://www.dicoding.com/academies/315/tutorials/16704?from=16703)

Sehingga, jika kita menjalankan kode JavaScript yang berjalan di luar browser environment, maka browser object (window) dan document object (document) tidak akan tersedia dan menyebabkan error. Sebagai contoh, jika kita menjalankan method alert()di environment browser, alert dialog akan muncul seperti berikut.

Namun, jika menggunakan [glot.io](https://glot.io/), method tersebut akan melemparkan error karena pada situs tersebut (compiler online) kode JavaScript dijalankan di environment NodeJS, yang mana tidak tersedianya **BOM** maupun **DOM**.



Keren, bukan? Dalam pemrograman front-end kita akan sering bergulat dengan BOM serta DOM untuk mempercantik tampilan website. Sudah siap untuk mengetahui lebih lanjut tentang keduanya?



Pada contoh yang kita pelajari di atas merupakan contoh yang sangat sederhana. Kita dapat melakukan lebih banyak hal lainnya, seperti mengubah konten elemen, memberikan event tertentu pada elemen, dan sebagainya.

Melalui contoh di atas, kita telah menggunakan console milik browser untuk menjalankan kode program JavaScript. Tentunya jika kita mengembangkan website tidak menggunakan pendekatan seperti ini. Ada cara lain dalam menulis kode JavaScript, yakni melalui berkas HTML secara langsung.

Di materi berikutnya, kita pelajari cara menyematkan kode JavaScript dengan dokumen HTML. Sudah penasaran, kan? Ayo kita menuju ke submodul berikutnya.

## Pendahuluan Browser Object Model (BOM)

Setiap browser menyediakan apa yang disebut Browser Object Model atau BOM yang bisa kita gunakan dalam kode JavaScript kita. Dalam browser, BOM ini diwakilkan oleh sebuah objek bernama window. Mari kita ketik window pada console browser dan lihat method dan method apa saja yang tersedia melalui objek ini:

Melalui objek window inilah kode JavaScript kita bisa mengakses berbagai method dan atribut yang bisa membantu kita membuat halaman web menjadi lebih interaktif. Kemungkinan Anda akan tertegun melihat begitu banyak method serta atribut. Akan tetapi, Anda tidak diharapkan untuk menghafal semuanya karena pada materi berikutnya kita cukup membahas 3 method paling dasar yakni alert(), prompt(), dan objek console.

## Anggota BOM

Pada sub-modul ini, kita akan membahas lebih dalam mengenai tiga method dari BOM. Ketiganya adalah alert(), prompt(), dan diakhiri dengan objek console.

### method alert()

Method alert() merupakan salah satu method BOM yang menerima satu parameter opsional berupa data string yang akan ditampilkan melalui sebuah pop-up browser. Contohnya jika kita ingin menampilkan sebuah pesan berisi string "Halo user!" menggunakan console browser, maka caranya adalah sebagai berikut:

1. alert('Halo user!');

Sehingga tampilannya akan terlihat seperti di bawah ini:

Sama seperti memanggil method pada umumnya pada JavaScript, bukan? Kita juga bisa memasukkan variabel sebagai parameternya seperit berikut.

1. let pesan = 'Halo user!';
2. alert(pesan);

Maka jika dijalankan pada console browser akan seperti berikut.

Keren, kan? Kita bisa menampilkan pesan yang telah ditentukan melalui JavaScript. Namun, ada satu hal yang perlu kita ingat saat menggunakan alert(), yakni ketika pop-up muncul, user tidak bisa berinteraksi dengan bagian lain dari halaman web sampai pop-up tersebut tersebut ditutup.

## Anggota BOM: prompt()

Sebelumnya kita telah mempelajari penggunaan method alert() untuk menampilkan pesan dalam dialog browser. Nah, selain menampilkan pesan, kita juga akan belajar cara mengambil data input dari user melalui dialog browser, yaitu prompt(). Method ini akan menampilkan sebuah dialog browser yang meminta user untuk mengisi kolom inputnya. Karena bisa menangkap input dari user, method ini memiliki lebih banyak fungsionalitas ketimbang alert(). Yuk, kita bahas satu-satu.

### Nilai Return dari prompt()

Jika kita lihat, sebenarnya method alert() mengembalikan sebuah value berupa undefined. Namun, prompt() akan mengembalikan sebuah nilai sesuai dengan inputkan oleh user. Sebagai contoh, mari jalankan kode berikut pada console browser.

1. let pesanInput = prompt('Masukkan pesan sesukamu...');

Lalu, coba kita tampilkan isi dari variabel pesanInput, maka hasilnya pasti akan sesuai dengan pesan yang kita input pada pop-up.

Kita bisa melihat bahwa nilai yang tersimpan di variabel pesanInput sesuai dengan input yang diberikan user, yakni "Aku suka belajar koding!". Jika user tidak mengisi apa pun dan tetap menekan tombol "OK", maka nilai yang diberikan hanya string kosong.

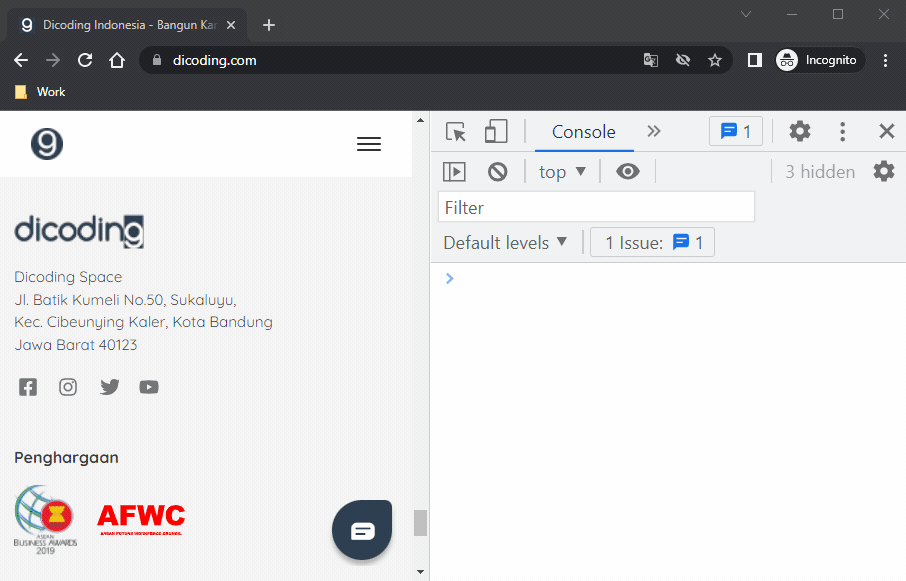
Pada contoh di atas, kita telah menekan tombol "OK", tetapi apa yang terjadi jika kita menekan tombol "Cancel"? Jika kita menekan tombol "Cancel", justru nilai yang dikembalikan adalah null. Silahkan coba kembali contoh di atas tetapi kali ini tekan tombol "Cancel", nanti nilai dari variabel pesanInput akan bernilai null.

Ada satu hal yang perlu kita ingat selama menggunakan prompt(), yakni nilai apa pun yang dimasukkan oleh user akan diproses dan dikembalikan menjadi data string. Silakan jalankan kode berikut pada console browser.

1. let pesanInput = prompt('Silakan masukkan angka...');

Ketika kita menampilkan tipe data dari variabel pesanInput, maka sudah pasti tipe datanya adalah string, walau yang kita masukkan berupa angka.

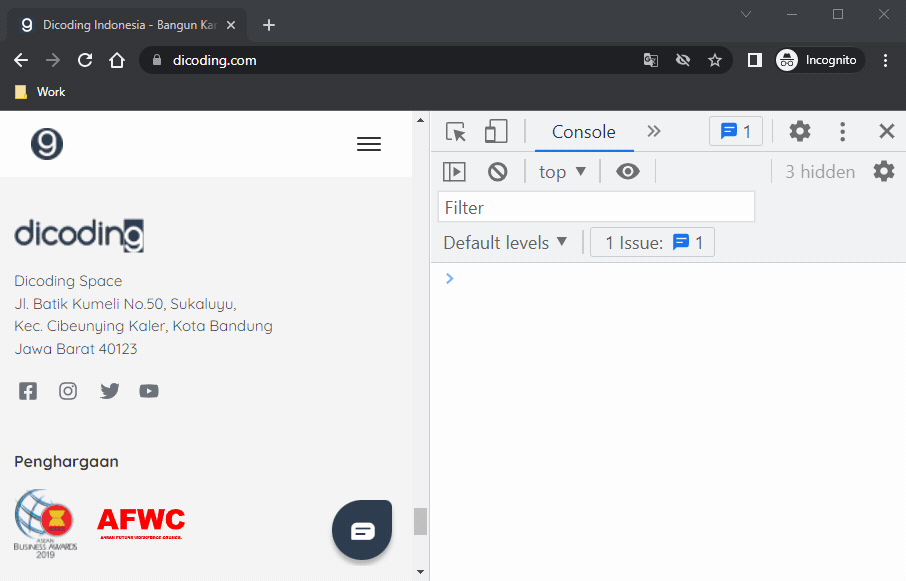
1. console.log(typeof pesanInput); // output -> 'string'

[](https://www.dicoding.com/academies/315/tutorials/16866)

Hal ini perlu kita ingat baik-baik. Kode tidak bisa menduga bahwa variabel pesanInput merupakan string atau number, ia akan tetap mengeluarkan string. Meskipun demikian, kita bisa cast atau parse hasilnya ke tipe data tertentu. Namun, harap lebih teliti karena jika kita salah melakukan parse ke tipe data maka akan menghasilkan NaN. Sebagai contoh, kita ingin mendapatkan value number dari prompt() maka kita bisa menggunakan cara ini.

1. let pesanInput = Number(prompt('Masukkan angka sesukamu...'));

Jika kita periksa tipe data dari hasilnya, maka hasilnya berupa number. Selain Number(), kita juga bisa menggunakan function parseInt() untuk melakukan parsing data string ke number.

[](https://www.dicoding.com/academies/315/tutorials/16866)

### Memberikan Nilai Default pada prompt()

Terdapat satu parameter lagi yang bisa kita masukkan ke prompt(), di mana parameter ini berguna sebagai nilai alternatif. Mari kita jalankan kode berikut.

1. let nama = prompt('Silakan masukkan nama Anda', 'Lorem Ipsum');

Ketika kita jalankan pada browser, maka kolom input yang muncul pada dialog box akan memiliki data secara otomatis.

Terdapat satu hal penting, yaitu ketika dialog box dari prompt() muncul, maka kita tidak bisa berinteraksi dengan komponen lain pada halaman web sampai dialog box tersebut ditutup. Hal ini sama halnya dengan alert().

**Anggota BOM: console**

Akhirnya kita sampai pada pembahasan objek *console* . Objek ini merupakan peralatan yang wajib diketahui oleh semua *Front-End* Web Developer. Mengapa demikian? Karena objek console memberikan kita akses ke fitur *debugging* alias peralatan-peralatan yang bisa membantu menghilangkan *bug* yang bersembunyi di dalam kode JavaScript.

Tentu Anda masih ingat jika kita ingin menampilkan sebuah nilai ke console browser, maka dapat menggunakan method log().

1. console.log('Pesan kamu');

Wah, ternyata kita sudah menggunakan objek *console* sejak awal! Melalui objek console ini, kita dapat membuat kode JavaScript menampilkan pesan-pesan khusus berdasarkan konteks tertentu pada *console* *browser*. Berikut beberapa *method* dari objek console yang umum digunakan:

| **Method** | **Penjelasan** |
| --- | --- |
| *log()* | Menampilkan pesan secara umum ke *console browser*. |
| *info()* | Menampilkan pesan yang berisi informasi penting ke *console browser.* |
| *warn()* | Menampilkan pesan yang berisi informasi dalam bentuk peringatan ke *console browser.* |
| *error()* | Menampilkan pesan yang berisi informasi dalam bentuk *error* ke *console browser*. |

## Cara Memanggil Anggota BOM

Sebelum kita lanjut ke materi berikutnya, kita perlu membahas satu hal terlebih dahulu. Ketika kalian ingat akan peraturan sintaks JavaScript, bukankah memanggil properti atau sebuah method dari suatu objek perlu menyertakan nama objek dan diikuti oleh nama properti atau functionnya? Hal ini benar dan berlaku untuk BOM yang diwakili oleh object window pada environment browser. Supaya lebih tergambarkan, perhatian dua baris kode ini.

1. // Cara pertama
2. window.alert('Hello World');
4. // Cara kedua
5. alert('Hello World');

Kedua cara di atas tidak memiliki perbedaan dan valid, yang mana cara pertama secara eksplisit memerintah kode JavaScript untuk memanggil method alert() milik objek window. Namun, cara yang kedua tidak menyebutkan objek window untuk menampilkan alert. Lalu, mengapa cara kedua tetap bisa dilakukan? Hal ini dikarenakan properti dan method yang dimiliki window bersifat global.

Walau cara kedua terkesan lebih singkat, kita harus tetap hati-hati karena jika pada scope sebuah berkas .js terdapat nama fungsi yang sama, maka pesan pada pop-up tidak akan muncul. Contohnya sebagai berikut:

1. function alert(nama) {
2. console.log('Hati-hati, ' + nama);
3. }
5. alert('Chewbacca'); // Output: Hati-hati, Chewbacca
6. // Output di atas akan tercetak ke *console browser*
8. window.alert('Chewbacca'); // Output: Hati-hati, Chewbacca
9. // Output di atas akan tetap tercetak pada console browser

Jadi harap hati-hati jika kita mendefinisikan sebuah method dengan nama yang sama dengan method milik window .

## Document Object Model (DOM)

Sejauh ini kita sudah mempelajari bagaimana cara menggunakan JavaScript untuk memanipulasi browser object melalui objek **window**. Namun, bagaimana jika kita ingin memanipulasi atau mengatur elemen-elemen yang terdapat pada berkas HTML dengan JavaScript? Jawabannya adalah bisa! Yakni melalui **Document Object Model**.

Apa sih **DOM**itu? Pada lingkungan browser, **DOM**memberikan izin bagi kode JavaScript untuk mengakses dan memanipulasi konten pada dokumen melalui sebuah Application Programming Interface (API), yakni sesuatu yang memungkinkan JavaScript dan dokumen HTML untuk "berkomunikasi". Masih ingat **Browser Object Model** yang diwakilkan oleh objek **window**yang bisa kita masukkan ke kode JavaScript? Nah, **DOM**diwakilkan oleh global objek bernama **document**, yang mana dapat dimanfaatkan oleh JavaScript.

Sebelum mempraktekkan penggunaan **DOM**dalam JavaScript, yuk kita pelajari bentuk struktur dari sebuah **DOM**.

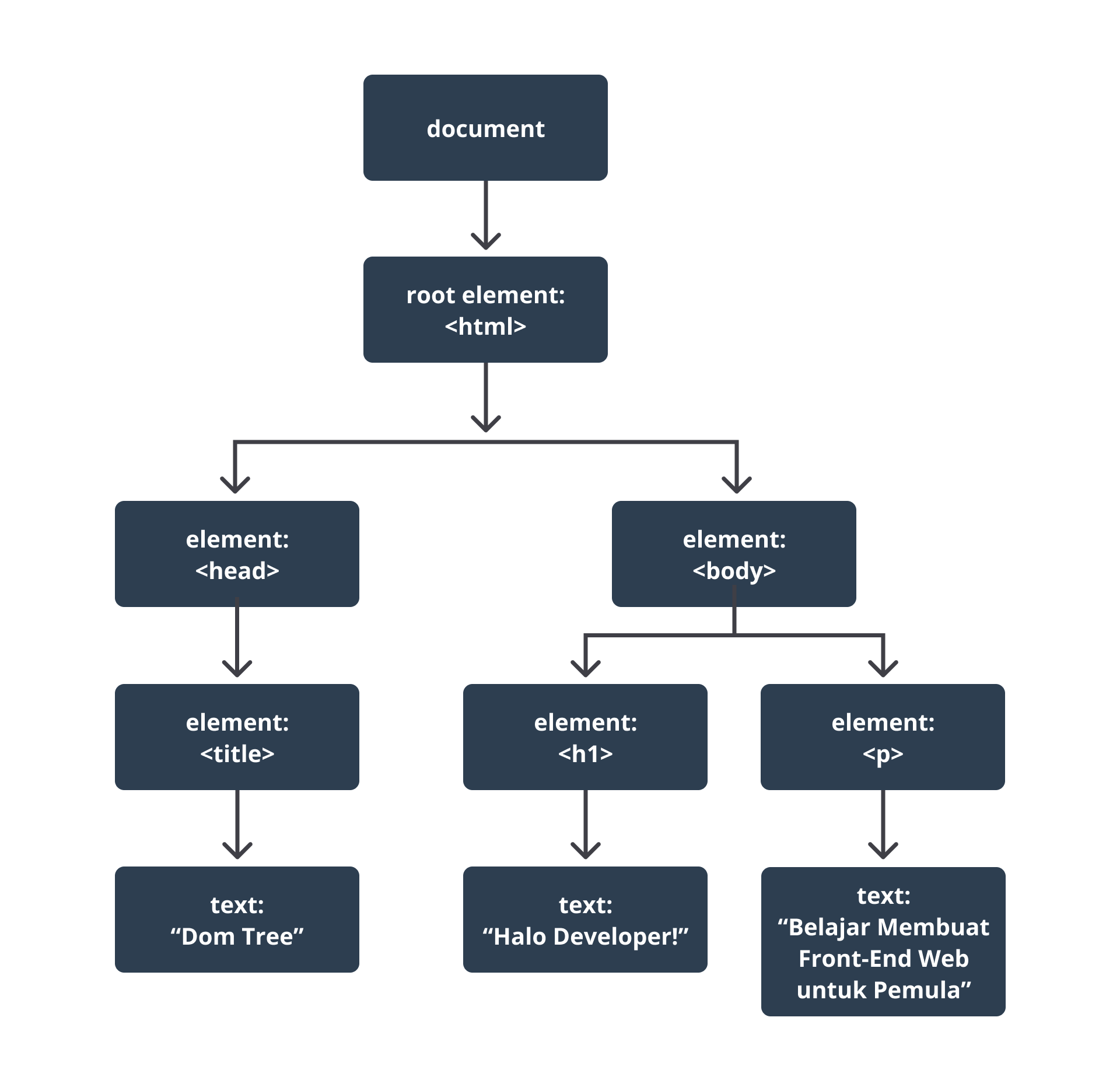
## DOM Tree

Ketika Anda membaca topik sub-modul ini, kemungkinan akan muncul pertanyaan “Apa hubungannya **DOM** dengan tree (pohon)?”. Tree yang dimaksud di sini adalah nama dari sebuah struktur data pada komputer yang secara visual mirip seperti sebuah pohon. Struktur data ini disebut tree, karena layaknya pohon terdapat satu batang induk tunggal yang kemudian bercabang menjadi batang-batang lainnya dan bisa saja bercabang kembali. Jika batang tersebut buntu, maka ujungnya terdapat daun (pada struktur data tree, daun disebut sebagai node).

Pada berkas HTML, batang induk tunggal adalah elemen <html>, sedangkan cabang-cabangnya adalah elemen-elemen yang terdapat di dalamnya. Misalkan kita mempunyai berkas HTML dengan struktur HTML berikut.

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <title>DOM Tree</title>
5. </head>
6. <body>
7. <h1>Hello Developer Front-End Web!</h1>
8. <p>Belajar Membuat Front-End Web untuk Pemula</p>
9. </body>
10. </html>

Jika kita buat berkas HTML di atas menjadi dalam bentuk **DOM**, strukturnya akan menjadi seperti berikut.

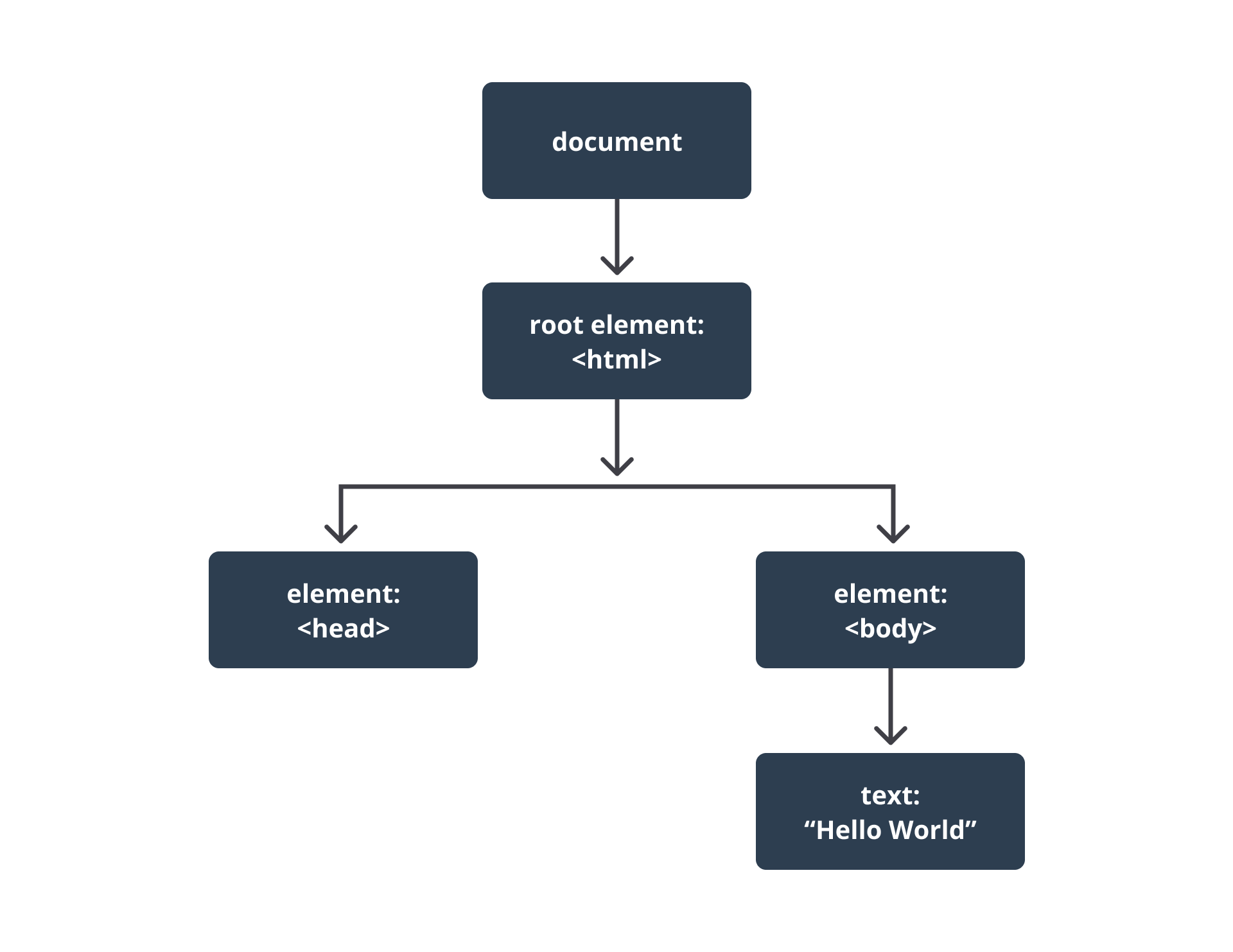
[](https://www.dicoding.com/academies/315/tutorials/16720?from=16719)

Pada contoh yang diberikan di atas, DOM memiliki bentuk struktur data tree yang dibuat berdasarkan berkas HTML di atas. Struktur data tree di atas inilah yang akan direpresentasikan dalam bentuk global obyek bernama **document**nanti.

Walaupun struktur dari DOM terbentuk berdasarkan isi dari berkas HTML, tetapi ada beberapa skenario tertentu yang menyebabkan struktur DOM berbeda dengan struktur elemen-elemen dalam berkas HTML. Salah satu skenarionya adalah jika terdapat kesalahan penulisan dalam berkas HTML. Mari kita lihat contohnya seperti berikut:

1. <!DOCTYPE html>
2. <html>
3. Hello World!
4. </html>

Pada berkas HTML di atas, jelas terlihat bahwa tidak ada elemen dengan tag <head> maupun <body> yang mengakibatkan berkas HTML tersebut tidak valid. Walaupun berkas HTML tersebut tidak valid pada bentuk DOM-nya, object structure-nya akan diperbaiki. Bagaimana caranya? Caranya yakni elemen dengan tag <head> maupun <body> akan ditambahkan secara otomatis kemudian teks “Hello World!” ditempatkan di elemen <body>.

[](https://www.dicoding.com/academies/315/tutorials/16720?from=16719)

**Mencari DOM (Mendapatkan DOM)**

Untuk mengakses sebuah elemen HTML melalui DOM, kita gunakan objek yang bernama **document*.***Objek ini berisi semua elemen yang berada pada HTML atau keseluruhan halaman yang terlihat pada jendela *browser* saat itu. Jika kita coba mengakses nilai dari global obyek **document** pada *console* *browser* maka tampilannya akan sebagai berikut:

Cukup dengan objek **document***,*kita dapat mengakses keseluruhan konten pada HTML yang terdapat root element <html>. Sehingga, jika dicermati kembali di dalamnya terdapat elemen lainnya seperti <head> dan <body>.

Secara sekilas isi dari objek **document** hanyalah terlihat seperti isi dari sebuah berkas HTML saja. Namun, layaknya sebuah objek JavaScript, objek **document** memiliki beberapa properti dan method yang bisa kita gunakan untuk mengakses koleksi elemen maupun elemen tertentu. Misalnya, mendapatkan konten-konten pada elemen dengan tag <head> melalui objek **document** dengan kode berikut:

1. const head = document.head;

Bagaimana jika kita ingin mendapatkan elemen <body> saja? Caranya hampir sama, yakni dengan kode berikut:

1. const body = document.body;

Kedua properti yakni *head* dan *body* akan mengembalikan elemen yang sama seperti *document* yakni sebuah objek.

Bagaimana jika kita ingin lebih spesifik lagi? Misalnya sebuah elemen tunggal? Kita bisa menggunakan *method-method* yang tersedia oleh objek **document**. Salah satu *method* yang bisa kita gunakan adalah getElementById()*.* Method tersebut berfungsiuntuk mendapatkan elemen berdasarkan nilai id*-*nya. Pada contoh di atas, elemen gambar  memiliki *id* "gambarUtama". Jika kita ingin mengakses elemen tersebut, caranya sebagai berikut:

1. const gambar = document.getElementById('gambarUtama');

Objek **document**juga memiliki banyak sekali *method* yang tersedia, berikut beberapa *method*yang umum digunakan serta fungsinya:

| **Method** | **Contoh** | **Fungsi** |
| --- | --- | --- |
| getElementById() | document.getElementById('display'); | Mengembalikan satu elemen yang memiliki nilai id "display". |
| getElementsByName() | document.getElementsByName('button') | Mengembalikan banyak elemen (HTMLCollection) yang memiliki attribute name dengan nilai "button". |
| getElementsByClassName() | document.getElementsByClassName('button') | Mengembalikan banyak elemen (HTMLCollection) yang memiliki attribute class dengan nilai "button". |
| getElementsByTagName() | document.getElementsByTagName('div') | Mengembalikan banyak elemen (HTMLCollection) yang merupakan <div> element. |
| querySelector() | document.querySelector('.button') | Mengembalikan elemen pertama (node) yang menerapkan class "button". |
| querySelectorAll() | document.querySelectorAll('.button') | Mengembalikan banyak Node dalam bentuk NodeList yang menerapkan class "button". |

Pada method di atas ada yang mengembalikan nilai HTML elemen secara tunggal. Selain itu, ada juga yang mengembalikan banyak elemen dari sebuah berkas HTML yang biasa disebut sebagai HTMLCollection. Dikarenakan semua *method*di atas dimiliki oleh objek document, maka jangan lupa mengawali semua pemanggilan *method-method* di atas dengan sintaks document.<nama\_method> ya.

Misalnya kita bisa mendapatkan elemen paling pertama yang menerapkan nilai "button" pada atribut *class* dengan *method* querySelector('.button').

Sedangkan untuk mendapatkan semua elemen yang nilai atribut *class*-nya "button", gunakan *method* querySelectorAll('.button'). *Method* tersebut akan mengembalikan semua elemen yang sesuai dalam bentuk HTMLCollection.

Ada fakta menarik tentang HTMLCollection, yakni memiliki karakteristik yang mirip dengan *array*. Contohnya, kita bisa menggunakan properti *length* untuk mendapatkan jumlah elemen yang terdapat di dalamnya. Selain itu, kita bisa mengakses nilai individual elemennya menggunakan *indexing*.

Karena HTMLCollection memiliki karakteristik yang mirip dengan *array*, maka kita juga bisa melakukan *looping* terhadap elemen-elemennya, yakni melalui sintaks *looping* *for of*. Berdasarkan contoh di atas, maka implementasinya sebagai berikut:

1. for (let item of buttons) {
2. console.log(item);
3. }

Sesuai dengan nama *method* tersebut, *method* querySelector() dan querySelectorAll() membutuhkan *query* khusus sebagai parameternya. Jika tujuan kita adalah mendapatkan elemen berdasarkan atribut *class*, maka parameternya harus diawali dengan tanda titik ("."), sedangkan jika berdasarkan atribut *id*, kita harus mengawali nilai parameternya dengan tanda pagar ("#"). Hal ini sama seperti ketika kita bekerja dengan selector di CSS.

Cobalah untuk bereksperimen dengan *method-method* yang sudah dijelaskan sebelumnya untuk mendapatkan pemahaman yang baik dan elemen yang sesuai. Kemudian carilah perbedaan dari fungsi *method-method*tersebut.