**Perkenalan JavaScript And DOM**

Pada sub-modul ini kita akan banyak berkenalan dengan JavaScript. Berbeda dengan HTML dan CSS, sekarang kita berada di wilayah pemrograman yang benar-benar memanfaatkan *logic* dalam penulisan kodenya. Pengembangan website tidak hanya mengandalkan seni untuk membangun tampilan yang cantik, melainkan juga memerlukan bahasa pemrograman agar meningkatkan fungsionalitas dan membuat website lebih interaktif.

Kita akan memulainya dengan memahami apa sebenarnya itu JavaScript, mencoba menulis dan menggunakannya pada sebuah website. Selanjutnya kita akan mengenal variabel, fungsi operator, perulangan dan hal lainnya. Kita juga akan belajar bagaimana memanipulasi elemen menggunakan syntax JavaScript, menampilkan alert dan mengetahui fungsi - fungsi dari Web API yang ada seperti *document*, *window* dan sebagainya.

Pada akhir sub-modul ini kita akan menerapkan apa yang sudah kita pelajari pada project Web Kalkulator sehingga kalkulator dapat berfungsi sesuai yang kita harapkan.

**Apa itu JavaScript?**

JavaScript merupakan bahasa pemrograman *client-side* sehingga seluruh prosesnya berjalan pada sisi pengguna bukan server. JavaScript diperlukan pada pengembangan website ketika kita membutuhkan suatu interaksi dari pengguna. Sesungguhnya website hanya menampilkan konten yang statis jika hanya menggunakan HTML dan CSS.

Karena diolah pada sisi *client*, JavaScript sangat bergantung pada pengaturan dan kemampuan browser ketika melakukan sebuah proses (compiling atau rendering pada DOM). Bahkan pengguna dapat sepenuhnya tidak mengizinkan JavaScript berjalan pada browser dengan menonaktifkan dukungan JavaScript pada browser.

Meskipun memiliki nama **Java**Script bahasa pemrograman ini sama sekali **tidak ada hubungannya dengan bahasa pemrograman Java**. Sebenarnya pada tahun 1995 Netscape melahirkan bahasa pemrograman ini dengan nama “LiveScript”, namun pada saat itu bahasa pemrograman Java sedang populer. Untuk memanfaatkan kepopulerannya, namanya pun diubah menjadi “JavaScript” dan benar bahasa pemrograman ini pun akhirnya memiliki popularitas yang tidak kalah dari Java.

Setelah diadopsi di luar Netscape, JavaScript distandarisasi oleh *European Computer Manufacturer’s Association* (ECMA) itulah sebabnya terkadang ada yang menyebutnya dengan ECMAScript.

Terdapat beberapa versi dari JavaScript. Pada tahun 2000 - 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 4 ditinggalkan.

Walaupun begitu, ini bukan akhir dari JavaScript. Pengembangan digantikan dengan ECMAScript 5 dengan mengurangi ambisinya dan hanya melakukan perbaikan pada hal yang tidak jadi kontroversi. Pembaharuan 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. Mulai dari itu tiap tahun JavaScript melakukan update bersifat minor.

## Menuliskan Syntax Javascript pada HTML

Sama seperti styling, untuk menggunakan JavaScript pada website kita bisa menerapkannya melalui atribut HTML, embed script, atau menggunakan file external.

### Atribut HTML

Untuk menuliskan JavaScript menggunakan atribut, kita bisa menerapkannya pada atribut event seperti “onclick”, sehingga JavaScript akan dieksekusi ketika elemen tersebut ditekan oleh kursor. Contohnya sebagai berikut:

1. <button onclick="alert('Anda menekan elemen button!')">Click di sini!</button>

Ada banyak sekali atribut event yang dapat digunakan untuk menuliskan script di dalamnya. Kita bisa lihat apa saja atributnya pada tautan berikut: <https://www.w3schools.com/tags/ref_eventattributes.asp>.  
  
Tentunya atribut tersebut kita gunakan sesuai dengan kebutuhan kita. onclick merupakan salah satu atribut yang common atau banyak digunakan karena interaksi tersebut sering pengguna lakukan.

### Embedded Script

JavaScript juga dapat dituliskan dengan menanamnya (embedding) pada berkas HTML dengan menggunakan elemen <script>.

1. <script>
2. *// JavaScript dituliskan di sini.*
3. </script>

Elemen <script> dapat diletakan di dalam elemen <head> atau <body>. Akan tetapi jika kita menerapkan banyak script pada elemen <head> proses memuat halaman akan menjadi lambat, karena HTML akan membaca kode dari atas ke bawah.

### External Script

Metode lainnya yaitu dengan menggunakan berkas external yang berekstensi **.js**. Di dalam berkas tersebutlah seluruh JavaScript dituliskan. Keuntungan menggunakan metode ini adalah script dapat diterapkan pada berbagai berkas HTML tanpa menuliskan ulang scriptnya (keuntungan yang sama juga ketika kita menggunakan external css).

Untuk menghubungkan external script dengan berkas HTML, kita gunakan elemen <script> lalu tambahkan atribut src dengan nilai alamat berkas **.js** yang digunakan.

1. <script src="berkas-javascript.js"></script>

Sama seperti Embedded Script, kita bisa tuliskan tag <script> tersebut di dalam elemen <head>. Namun, direkomendasikan untuk disimpan di dalam elemen <body> sebelum tag penutup </body>.

Teknik inilah yang akan kita gunakan dalam penerapan JavaScript pada project kalkulator.

## Membuat Berkas JavaScript Pertama

Setelah mengetahui beberapa metode untuk menuliskan JavaScript pada HTML, mari kita terapkan salah satu metode yang sudah kita ketahui. Kita akan menerapkan metode external script dengan membuat berkas JavaScript yang nantinya akan digunakan untuk menuliskan script dari project kalkulator yang sudah kita buat.

Buka project kalkulator yang sudah kita buat. Di dalam folder **assets**, buat berkas baru dengan memberikan nama “**kalkulator.js**”. Ingat berkas ini harus diakhiri dengan ekstensi **.js**. Ekstensi tersebut menandakan bahwa berkas ini merupakan berkas JavaScript.

Setelah membuat berkas tersebut maka struktur dari proyek kalkulator akan tampak seperti berikut:

Buka berkas **kalkulator.js** dan tuliskan kode JavaScript berikut:

1. console.log("Selamat Anda berhasil menggunakan JavaScript pada Website");

Jangan khawatir jika Anda belum memahami maksud dari kode tersebut, nanti akan dijelaskan apa yang sudah kita tuliskan.  
  
Simpan perubahan pada berkas JavaScript tersebut kemudian buka **index.html** dan tambahkan elemen <script> dengan atribut src bernilai alamat berkas **kalkulator.js**. Tuliskan elemen <script> tepat sebelum closing tag </body>.

1. ...........
2. <script src="assets/kalkulator.js"></script>
3. </body>

Simpan perubahan pada **index.html** kemudian kita coba buka halaman tersebut dengan menggunakan browser. Maka hasilnya akan nampak seperti ini.

Tidak ada perubahan apa apa bukan? Yups, benar. Tetapi ketika kita mencoba membuka console pada DevTools kita akan mendapati pesan yang dituliskan menggunakan JavaScript. Silakan buka console pada DevTools dengan menekan ctrl+shift+i.

Script yang kita tuliskan sebelumnya berfungsi untuk menampilkan sebuah data baik itu teks (string) atau variabel, objek, fungsi dsb. Pada console website  console.log() biasanya digunakan sebagai sarana debugging sederhana untuk mengetahui nilai dari suatu variabel.

Mungkin ini akan membuat Anda bingung jika tidak memiliki dasar pemrograman. Tak apa, yang terpenting ketika pesan ini muncul pada console, itu menandakan kita sudah berhasil menerapkan JavaScript pada website kita.

**Dasar JavaScript**

Wajar jika kita menemukan banyak buku JavaScript dengan halaman super tebal karena memang banyak sekali bahasan seputar JavaScript ini. Tetapi pada materi ini kita rangkum menjadi beberapa pembahasan penting yang Anda perlu ketahui. Tujuannya agar Anda familier dengan dasar - dasar JavaScript. Dengan memahami dasar-dasar JavaScript, nantinya diharapkan kita dapat terus berlatih sehingga terbiasa dengan penggunaan sintaksnya.

Ketika mempelajari bahasa pemrograman apapun pertama kita harus memahami logika yang mendasari dari penulisan *script*, meskipun secara sintaksis setiap bahasa pemrograman berbeda namun secara umum logikanya akan sama.

Maka dari itu hindari belajar dengan cara menghafal seluruh struktur kode yang dituliskan, tapi belajarlah dengan memahami tujuan dari kode yang dituliskan. Dengan memahami kode-kode yang ada dan dipadukan logika maka kita akan mudah dalam menghadapi suatu masalah - masalah yang ada ketika mengembangkan suatu aplikasi/web.

Pada akhir materi ini, kita akan mengenal tentang variabel, array, operator komparasi, *if else statement* dan masih banyak lagi. Sudah siap belajar?

Mungkin selama pembelajaran dasar JavaScript kita perlu mencoba coba kode yang sedang dipelajari, kami menyediakan interaktif kode yang bisa Anda jalankan dan ubah secara langsung. Selain itu Anda bisa gunakan tools online seperti [glot.io](https://glot.io/) ketika menjalankan kodenya. Atau Anda juga bisa mencobanya langsung pada console browser melalui DevTools. Silakan baca artikel berikut sebelum melanjutkan pada materi selanjutnya:

* [Glot.io: Code Editor dan Compiler Online Terbaik](https://www.dicoding.com/blog/code-editor-dan-compiler-online-terbaik/)
* [Get Started With Running JavaScript In The Console](https://developers.google.com/web/tools/chrome-devtools/console/javascript)

### Statement

Sebuah script dibangun dari serangkaian statement. Statement merupakan sebuah perintah yang bertujuan untuk memberitahu apa yang harus dilakukan browser. Contohnya kode berikut merupakan statement yang menyatakan bahwa browser harus menampilkan pesan (alert) dengan kalimat “Terima kasih”.

1. alert("Terimakasih.");

Pada akhir kode terdapat tanda titik koma (;) yang digunakan untuk menandai akhir dari sebuah statement. Meskipun sebenarnya pada JavaScript kita dapat menghiraukan penulisan titik koma (;) pada akhir statement, tetapi dalam best practice nya kita biasakan akhiri statement dengan titik koma (;).

### Comment

Pada JavaScript juga terdapat fitur komentar. Seluruh teks yang dituliskan pada tanda komentar akan diabaikan (tidak dianggap ada) atau tidak akan dieksekusi. Dalam komentar ini kita bisa menuliskan teks untuk mengingatkan atau menjelaskan kode yang kita tuliskan. Hal ini bisa saja berguna jika kode yang kita tuliskan akan diubah oleh orang lain. Tapi ingat, jangan terlalu berupaya dalam menuliskan komentar yang sebenarnya tidak perlu dituliskan.

Terdapat dua metode ketika memberikan komentar. Untuk memberikan komentar pada satu baris saja, kita bisa gunakan tanda dua buah garis miring (//) pada awal barisnya.

1. *// ini merupakan komentar satu baris komentar*
2. *// alert("Terima kasih")*

Sedangkan untuk menggunakan komentar lebih dari satu baris (multiple lines comment) kita bisa gunakan tanda /\* sebagai pembuka komentar dan tanda \*/ sebagai penutup komentar. Teks apapun yang berada di antara tanda tersebut akan dijadikan komentar dan tidak akan dieksekusi.

1. */\* Ini merupakan komentar dengan lebih dari satu baris*
2. *Teks apapun yang berada disini akan dijadikan komentar.*
3. *Ketika menggunakan ini, jangan lupa untuk menutupnya.*
4. *\*/*

### Variabel

Ketika mendengar variabel apakah Anda teringat tentang pelajaran matematika? Yups, variabel di sini memiliki konsep yang sama. Variabel umumnya digunakan untuk menyimpan informasi atau nilai yang akan dikelola di dalam sebuah program.

Sebelum ECMAScript 2015 (ES6) untuk membuat variabel pada JavaScript kita gunakan keyword var**.**

1. var firstName = "Harry";

Tanda sama dengan (=) digunakan untuk menginisialisasikan nilai pada variabel, sehingga sekarang variabel **firstName**memiliki nilai teks **“Harry”**.

Kita bisa menggunakan apapun yang kita mau untuk menamai sebuah variabel, tetapi pastikan penamaannya masih masuk akal dengan konteksnya agar berikutnya 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 dapat mengandung spasi (white space), jika penamaan variabel lebih dari dua kata maka tuliskan secara camelCase. Contoh firstName, lastName, catName, dll.
* Tidak dapat mengandung karakter spesial (! . , / \ + \* = dll.)

Nilai variabel yang diinisialisasi menggunakan var dapat diubah kembali nilainya, contoh:

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5790?from=5789#run1-editor1)



1

var firstName = "Harry";

2

console.log(firstName);

3

​

4

firstName = "Ron";

5

console.log(firstName);

6

​

7

/\* output:

8

Harry

9

Ron

10

\*/

 Input  Reset

 Jalankan

Harry Ron

Sejak ECMAScript 2015 (ES6) selain var, menginisialisasikan variabel dapat menggunakan **let** dan **const**. ES6 melakukan improvisasi pada deklarasi variabel karena menggunakan var terdapat beberapa hal yang kontroversial, salah satunya hoisting.

Apa itu hoisting? sesuai artinya “Mengangkat” variabel yang dideklarasikan menggunakan var ini dapat diberikan nilai terlebih dahulu sebelum dideklarasikan, Contoh:

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5790?from=5789#run2-editor1)



1

x = 100;

2

var x;

3

console.log(x);

4

​

5

/\* output: 100 \*/

 Input  Reset

 Jalankan

100

Ini dikarenakan proses hoisting, sebenarnya di belakang layar deklarasi variabel x diangkat ke atas sehingga kode yang tampak seperti ini:

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5790?from=5789#run3-editor1)



1

var x;

2

x = 100;

3

console.log(x);

4

​

5

/\* output: 100 \*/

 Input  Reset

 Jalankan

100

Hoisting menjadi kontroversial karena tidak sedikit developer yang dibuat bingung akan hal ini. Masalah ini sudah terselesaikan jika kita menggunakan let dalam mendeklarasikan variabel.

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5790?from=5789#run4-editor1)



1

y = 100;

2

let y;

3

console.log(y);

4

​

5

/\* ReferenceError: Cannot access 'y' before initialization \*/

 Input  Reset

 Jalankan

100

Kemudian **const** digunakan untuk mendeklarasikan sebuah variabel yang sifatnya immutable atau tidak perlu diubah nilainya. Jika kita menginisialisasi kembali nilai variabel yang menggunakan const, maka akan mendapati eror “TypeError: Assignment to constant variable.”

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5790?from=5789#run5-editor1)



1

const z = 100;

2

console.log(z);

3

​

4

z = 200;

5

console.log(z)

6

​

7

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

 Input  Reset

 Jalankan

/home/glot/main.js:4 z = 200; ^ TypeError: Assignment to constant variable. at Object.<anonymous> (/home/glot/main.js:4:3) at Module.\_compile (node:internal/modules/cjs/loader:1103:14) at Object.Module.\_extensions..js (node:internal/modules/cjs/loader:1155:10) at Module.load (node:internal/modules/cjs/loader:981:32) at Function.Module.\_load (node:internal/modules/cjs/loader:822:12) at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run\_main:77:12) at node:internal/main/run\_main\_module:17:47

Dengan begitu sebaiknya kita gunakan let atau const ketika mendeklarasikan variabel daripada menggunakan var.

### Variabel

Ketika mendengar variabel apakah Anda teringat tentang pelajaran matematika? Yups, variabel di sini memiliki konsep yang sama. Variabel umumnya digunakan untuk menyimpan informasi atau nilai yang akan dikelola di dalam sebuah program.

Sebelum ECMAScript 2015 (ES6) untuk membuat variabel pada JavaScript kita gunakan keyword var**.**

1. var firstName = "Harry";

Tanda sama dengan (=) digunakan untuk menginisialisasikan nilai pada variabel, sehingga sekarang variabel **firstName**memiliki nilai teks **“Harry”**.

Kita bisa menggunakan apapun yang kita mau untuk menamai sebuah variabel, tetapi pastikan penamaannya masih masuk akal dengan konteksnya agar berikutnya 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 dapat mengandung spasi (white space), jika penamaan variabel lebih dari dua kata maka tuliskan secara camelCase. Contoh firstName, lastName, catName, dll.
* Tidak dapat mengandung karakter spesial (! . , / \ + \* = dll.)

Nilai variabel yang diinisialisasi menggunakan var dapat diubah kembali nilainya, contoh:

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5790#run1-editor1)



1

var firstName = "Harry";

2

console.log(firstName);

3

​

4

firstName = "Ron";

5

console.log(firstName);

6

​

7

/\* output:

8

Harry

9

Ron

10

\*/

 Input  Reset

 Jalankan

Sejak ECMAScript 2015 (ES6) selain var, menginisialisasikan variabel dapat menggunakan **let** dan **const**. ES6 melakukan improvisasi pada deklarasi variabel karena menggunakan var terdapat beberapa hal yang kontroversial, salah satunya hoisting.

Apa itu hoisting? sesuai artinya “Mengangkat” variabel yang dideklarasikan menggunakan var ini dapat diberikan nilai terlebih dahulu sebelum dideklarasikan, Contoh:

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5790#run2-editor1)



1

x = 100;

2

var x;

3

console.log(x);

4

​

5

/\* output: 100 \*/

 Input  Reset

 Jalankan

Ini dikarenakan proses hoisting, sebenarnya di belakang layar deklarasi variabel x diangkat ke atas sehingga kode yang tampak seperti ini:

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5790#run3-editor1)



1

var x;

2

x = 100;

3

console.log(x);

4

​

5

/\* output: 100 \*/

 Input  Reset

 Jalankan

Hoisting menjadi kontroversial karena tidak sedikit developer yang dibuat bingung akan hal ini. Masalah ini sudah terselesaikan jika kita menggunakan let dalam mendeklarasikan variabel.

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5790#run4-editor1)



1

y = 100;

2

let y;

3

console.log(y);

4

​

5

/\* ReferenceError: Cannot access 'y' before initialization \*/

 Input  Reset

 Jalankan

Kemudian **const** digunakan untuk mendeklarasikan sebuah variabel yang sifatnya immutable atau tidak perlu diubah nilainya. Jika kita menginisialisasi kembali nilai variabel yang menggunakan const, maka akan mendapati eror “TypeError: Assignment to constant variable.”

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5790#run5-editor1)



1

const z = 100;

2

console.log(z);

3

​

4

z = 200;

5

console.log(z)

6

​

7

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

 Input  Reset

 Jalankan

Dengan begitu sebaiknya kita gunakan let atau const ketika mendeklarasikan variabel daripada menggunakan var.

### ipe Data

Nilai yang kita tetapkan pada variabel pasti memiliki tipe data. Tipe data merupakan pengklasifikasian data berdasarkan jenis data tersebut. Pada JavaScript terdapat beberapa tipe data sebagai berikut:

#### Undefined

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

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5793?from=5790#run1-editor1)



1

let x;

2

console.log(typeof(x));

3

​

4

/\* output: undefined \*/

 Input  Reset

 Jalankan

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

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. JavaScript variabel bertipe data number dituliskan seperti ini:

1. let x = 10;

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

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5793?from=5790#run2-editor1)



1

let x = 10;

2

console.log(typeof(x))

3

​

4

/\* output: number \*/

5

​

6

let y = 17.25;

7

console.log(typeof(y))

8

​

9

/\* output: number \*/

 Input  Reset

 Jalankan

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

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

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5793?from=5790#run3-editor1)



1

let a = 12;

2

let b = 9;

3

​

4

console.log(a + b)

5

console.log(a - b)

6

console.log(a \* b)

7

console.log(a / b)

8

console.log(a % b)

9

​

10

/\* output:

11

21

12

3

13

108

14

1.3333333333333333

15

3

16

\*/

 Input  Reset

 Jalankan

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

Lebih jelasnya berikut contoh kode dalam penerapan operator tersebut, perhatikan hasil yang di dapat.

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5793?from=5790#run4-editor1)



1

/\* Increment dan Decrement \*/

2

​

3

let postfix = 5;

4

console.log(postfix++);

5

/\* output: 5 \*/

6

​

7

let prefix = 5;

8

console.log(++prefix);

9

/\* output: 6 \*/

 Input  Reset

 Jalankan

#### Strings

Tipe data selanjutnya adalah strings, String ini dasarnya adalah sebuah teks. Pada JavaScript untuk menetapkan nilai string pada variabel, gunakan tanda single (‘) atau double quote (“) di antara teksnya. Contohnya:

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5793?from=5790#run5-editor1)



1

let greet = "Hello";

2

console.log(typeof(greet))

3

​

4

/\* output: string \*/

 Input  Reset

 Jalankan

Apa pun yang berada di antara tanda double quote atau single quote merupakan sebuah string, baik itu angka, huruf maupun karakter spesial.

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

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5793?from=5790#run6-editor1)



1

let greet = "Hello";

2

let moreGreet = greet + greet;

3

console.log(moreGreet);

4

​

5

/\* output: HelloHello \*/

 Input  Reset

 Jalankan

#### Boolean

Boolean hanya dapat memiliki dua nilai, yakni true atau false. Tipe data ini menjadi kunci utama dalam penentuan logika, kita akan memahaminya nanti ketika pembahasan if/else statement. Untuk menetapkan nilai boolean pada variabel kita bisa menggunakan keyword true atau false.

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5793?from=5790#run7-editor1)



1

let x = true;

2

let y = false;

3

​

4

console.log(typeof(x))

5

console.log(typeof(y))

6

​

7

/\* output:

8

boolean

9

boolean

10

\*/

 Input  Reset

 Jalankan

Atau kita bisa gunakan operator komparasi seperti lebih dari (>) atau kurang dari (<). Contohnya:

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5793?from=5790#run8-editor1)



1

const a = 10;

2

const b = 12;

3

​

4

let isGreater = a > b;

5

let isLess = a < b;

6

​

7

console.log(isGreater);

8

console.log(isLess);

9

​

10

/\* output:

11

false

12

true

13

\*/

 Input  Reset

 Jalankan

#### Null

Yang terakhir 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 apapun. Nah, daripada kita tidak menetapkan nilai apapun (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 diinisialisasikan.

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5793?from=5790#run9-editor1)



1

let someLaterData = null;

2

console.log(someLaterData);

3

​

4

/\* output:

5

null

6

\*/

 Input  Reset

 Jalankan

Meskipun terdapat beberapa tipe data dalam JavaScript, tetapi variabel pada JavaScript memiliki sifat tipe data yang dinamis. Artinya, kita dapat memberikan tipe data yang berubah-ubah pada satu variabel yang sama. Contohnya:

1. let x; *// x merupakan undefined*
2. x = 1 *// sekarang x merupakan number*
3. x = true *// sekarang x merupakan boolean*
4. x = "Harry" *// sekarang x merupakan string*

### Arrays dan Objek

Kita sudah mengenal tipe data dasar yang ada pada JavaScript. Kali ini kita akan membahas mengenai array dan objek pada JavaScript. Keduanya dapat menampung lebih dari satu tipe data dasar yang digunakan untuk mengelola sebuah data.

#### Arrays

Array merupakan tipe data yang dapat mengelompokkan lebih dari satu nilai dari tipe data lain dengan menempatkannya pada satu variabel. Contoh:

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5798?from=5793#run1-editor1)



1

let myArray = ["Coklat", 42.5, 22, true, "Programming"];

2

console.log(myArray);

3

​

4

/\* output:

5

[ 'Coklat', 42.5, 22, true, 'Programming' ]

6

\*/

 Input  Reset

 Jalankan

[ 'Coklat', 42.5, 22, true, 'Programming' ]

Nilai - nilai yang berada pada array disusun dan diakses secara indexing. Untuk mengakses nilai di dalam array kita gunakan tanda kurung siku [] yang di dalamnya berupa angka yang merupakan posisi nilai yang ingin diakses.

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5798?from=5793#run2-editor1)



1

let myArray = ["Coklat", 42.5, 22, true, "Programming"];

2

console.log(myArray[1]);

3

​

4

/\* output:

5

42.5

6

\*/

 Input  Reset

 Jalankan

42.5

Yang perlu kita ketahui adalah nilai index dimulai dari angka 0. Terlihat pada contoh kode di atas, kita mengakses index ke-1 pada myArray. Nilai yang muncul adalah nilai ke-2 dari array tersebut, yakni 42.5. Jika kita mengakses nilai array lebih dari index-nya maka hasilnya akan undefined. Index terakhir array selalu jumlah nilai array - 1.

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5798?from=5793#run3-editor1)



1

let myArray = ["Coklat", 42.5, 22, true, "Programming"];

2

console.log(myArray[0]);

3

console.log(myArray[1]);

4

console.log(myArray[2]);

5

console.log(myArray[3]);

6

console.log(myArray[4]);

7

console.log(myArray[5]);

8

console.log("Panjang nilai myArray adalah " + myArray.length + ".");

9

​

10

/\* output:

11

Coklat

12

42.5

13

22

14

true

15

Programming

16

undefined

17

Panjang nilai myArray adalah 5.

18

\*/

 Input  Reset

 Jalankan

Coklat 42.5 22 true Programming undefined Panjang nilai myArray adalah 5.

#### Objek

Objek serupa dengan array yang dapat menampung banyak nilai dengan tipe data yang beragam. Untuk mengelola data menggunakan objek, bedanya objek diakses tidak melalui indexing,  melainkan menggunakan pendekatan key-value. Untuk mengakses nilainya kita gunakan key. Key juga biasa disebut dengan properti.

Untuk menetapkan objek pada variabel gunakan tanda kurung kurawal { } dalam menginisialisasinya. Kemudian di dalamnya kita tetapkan **key: value**.

1. let object = {key1: "value1", key2: "value2", key3: "value3"}

Dalam menentukan nama key, gunakanlah nama yang dapat mendeskripsikan dari value-nya. Pada value, kita dapat mengisikan nilai dengan tipe data apapun, termasuk array. Contoh:

1. let user = {firstName: "Harry", lastName: "Potter",  age: 20, isMuggle: false, stuff: ["Wand", "Flying Car", "Owl"]};

Dalam menuliskan objek, baris baru tidaklah penting dan tidak akan berpengaruh apa pun. Sehingga lebih baik setiap kita menetapkan key-value buatlah baris baru untuk memisahkan antar nilainya, hal ini akan memudahkan kita dalam memahami struktur data yang berada pada objek.

1. let user = {
2. firstName: "Harry",
3. lastName: "Potter",
4. age: 20,
5. isMuggle: false,
6. stuff: ["Magic Wind", "Flying Car", "Owl"]
7. };

Kemudian untuk mengakses nilai dari properti objek kita dapat gunakan tanda titik diikuti dengan nama properti-nya. Contoh:

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5798?from=5793#run4-editor1)



1

let user = {

2

firstName: "Harry",

3

lastName: "Potter",

4

age: 20,

5

isMuggle: false,

6

stuff: ["Magic Wind", "Flying Car", "Owl"]

7

};

8

​

9

console.log("Hallo, nama saya " + user.firstName + " " + user.lastName);

10

console.log("Umur saya " + user.age + " tahun");

11

​

12

/\* output

13

Hallo, nama saya Harry Potter

14

Umur saya 20 tahun

15

\*/

 Input  Reset

 Jalankan

Bahkan dalam properti objek, kita dapat menyimpan nilai objek lainnya. Contohnya properti firstName dan lastName dapat dikelompokan kembali dalam sebuah objek baru sebagai berikut:

1. let user = {
2. name: {
3. first: "Harry",
4. last: "Potter"
5. },
6. age: 20,
7. isMuggle: false,
8. stuff: ["Magic Wind", "Flying Car", "Owl"]
9. }

Untuk mendapatkan nilainya kita perlu mengakses properti dari objek user kemudian name. Sehingga penulisannya menjadi seperti berikut:

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5798?from=5793#run5-editor1)



1

let user = {

2

name: {

3

first: "Harry",

4

last: "Potter",

5

},

6

age: 20,

7

isMuggle: false,

8

stuff: ["Magic Wind", "Flying Car", "Owl"]

9

};

10

​

11

console.log("Hallo, nama saya " + user.name.first + " " + user.name.last);

12

console.log("Umur saya " + user.age + " tahun");

13

​

14

/\* output

15

Hallo, nama saya Harry Potter

16

Umur saya 20 tahun

17

\*/

 Input  Reset

 Jalankan

Mungkin seperti itulah gambaran mengenai objek di JavaScript, penting untuk Anda garis bawahi bahwa dalam mengelola banyak tipe data pada JavaScript, Anda dapat menggunakan array maupun objek. Pembahasan di atas cukup sebagai landasan pengetahuan mengenai dua hal itu ya.  
  
Banyak hal sebenarnya yang dapat diceritakan tentang dua hal ini, terutama untuk objek. Jika Anda ingin tahu lebih dalam, Anda bisa baca dokumentasinya pada tautan yang disediakan oleh MDN:

* Array : <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array>
* Objek : <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object>

### Assignment Operator

Dari contoh kode yang kita gunakan sebelumnya sebenarnya kita sudah menggunakan assignment operator. Operator ini digunakan untuk memberikan nilai pada variabel.

Pada dasarnya operator ini adalah tanda sama dengan (=), di mana tanda ini digunakan untuk menginisialisasi nilai pada variabel. Variabel yang akan diberikan nilai ditempatkan pada sebelah kiri dan nilainya ditempatkan pada sebelah kanan (nilai dapat berupa variabel lain atau nilai primitif, array, atau objek). Di antara keduanya terdapat operator assignment.

1. x = y;

Contoh pada statement tersebut kita menginisialisasikan nilai y pada variabel x, sehingga nilai x sekarang memiliki nilai yang sama dengan y.

Ada beberapa assignment operator tambahan lain dalam menginisialisasikan nilai pada variabel, atau bisa kita sebut sebagai shortcut dalam menentukan nilai. Contohnya:

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/6114?from=5798#run1-editor1)



1

let x = 10;

2

let y = 5

3

​

4

x += y;

5

​

6

console.log(x);

7

​

8

/\* output

9

15

10

\*/

 Input  Reset

 Jalankan

15

Pada contoh kode di atas bisa terdapat statement x += y; apa itu artinya? Assignment operator tersebut digunakan sebagai shortcut dari x = x + y. Cara ini juga dapat digunakan pada operator aritmatika lain seperti, perkalian, pengurangan, pembagian, dan lainnya.

1. let x = 10;
2. let y = 5
4. x += y; // artinya -> x = x + y;
5. x -= y; // artinya -> x = x - y;
6. x \*= y; // artinya -> x = x \* y;
7. x /= y; // artinya -> x = x / y;
8. x %= y; // artinya -> x = x % y;

### Operator Komparasi

Sekarang kita sudah mengetahui bagaimana cara menyimpan nilai pada sebuah variabel, array, ataupun objek. Nah, selanjutnya kita akan belajar mengenai operator komparasi sebagai logika dasar dalam membandingkan nilai pada JavaScript.

Terdapat serangkaian karakter khusus yang disebut dengan operator pembanding/komparasi yang dapat mengevaluasi dan membandingkan dua nilai. Berikut daftar operator dan fungsinya:

| **Operator** | **Fungsi** |
| --- | --- |
| == | Membandingkan kedua nilai **apakah sama.** (Tidak Identik) |
| != | Membandingkan kedua nilai **apakah tidak sama**. (Tidak Identik) |
| === | Membandingkan kedua nilai **apakah identik**. |
| !== | Membandingkan kedua nilai **apakah tidak identik.** |
| > | Membandingkan dua nilai **apakah nilai pertama lebih besar dari nilai kedua**. |
| >= | Membandingkan dua nilai **apakah nilai pertama lebih besar atau sama dengan dari nilai kedua**. |
| < | Membandingkan dua nilai apakah nilai pertama **lebih kecil dari nilai kedua.** |
| <= | Membandingkan dua nilai **apakah nilai pertama lebih kecil dari atau sama dengan nilai kedua.** |

Ketika kita melakukan perbandingan antara dua nilai, JavaScript akan mengevaluasi kedua nilai tersebut dan akan mengembalikan boolean dengan nilai hasil perbandingan tersebut, baik false, atau true. Berikut contohnya:

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5799?from=6114#run1-editor1)



1

let a = 10;

2

let b = 12;

3

​

4

console.log(a < b);

5

console.log(a > b);

6

​

7

/\* output

8

true

9

false

10

\*/

 Input  Reset

 Jalankan

true false

#### Perbedaan antara “sama” dan “Identik”

Dalam operator komparasi di JavaScript, hal yang menjadi sedikit “tricky” adalah membedakan antara “sama” (==) dan “identik” (===).

Kita sudah mengetahui bahwa setiap nilai pasti memiliki tipe data baik itu number, string atau boolean. Contohnya sebuah string “10” dan number 10 merupakan hal yang serupa, tetapi keduanya tidak benar-benar sama.

Hal inilah yang membedakan antara sama dan identik pada JavaScript. Jika kita ingin membandingkan hanya dari kesamaan nilainya kita bisa gunakan == tapi jika kita ingin membandingkan dengan memperhatikan tipe datanya kita gunakan ===.

Contohnya sebagai berikut:

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5799?from=6114#run2-editor1)



1

const aString = '10';

2

const aNumber = 10

3

​

4

console.log(aString == aNumber) // true, karena nilainya sama-sama 10

5

console.log(aString === aNumber) // false, karena walaupun nilainya sama, tetapi tipe datanya berbeda

6

​

7

/\* output

8

true

9

false

10

\*/

 Input  Reset

 Jalankan

true false

#### Logical Operators

Terdapat beberapa operator lain yang dapat kita gunakan untuk menetapkan logika yang lebih kompleks, yakni dengan logical operators. Dengan logical operator kita dapat menggunakan kombinasi dari dua nilai boolean atau bahkan lebih dalam menetapkan logika.

Pada JavaScript terdapat tiga buah karakter khusus yang berfungsi sebagai logical operator, berikut macam-macam logical operator dan fungsinya:

| **Operator** | **Deskripsi** |
| --- | --- |
| && | Operator dan (and), logika akan menghasilkan **true** **apabila** **semua kondisi terpenuhi** (bernilai true). |
| || | Operator atau (or), logika akan menghasilkan **true** **apabila ada** **salah satu kondisi terpenuhi** (bernilai true). |
| ! | Operator tidak (not), digunakan untuk membalikan suatu kondisi. |

Berikut contoh penerapannya pada JavaScript:

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5799?from=6114#run3-editor1)



1

let a = 10;

2

let b = 12;

3

​

4

/\* AND operator \*/

5

console.log(a < 15 && b > 10); // (true && true) -> true

6

console.log(a > 15 && b > 10); // (false && true) -> false

7

​

8

/\* OR operator \*/

9

console.log(a < 15 || b > 10); // (true || true) -> true

10

console.log(a > 15 || b > 10); // (false || true) -> true

11

​

12

/\* NOT operator \*/

13

console.log(!(a < 15)); // !(true) -> false

14

console.log(!(a < 15 && b > 10)); // !(true && true) -> !(true) -> false

15

​

16

/\* output

17

true

18

false

19

true

20

true

21

false

22

false

23

\*/

 Input  Reset

 Jalankan

true false true true false false

Mungkin sebagian dari kita bertanya, sebenarnya apa kegunaan dari nilai boolean selain hanya menampilkan nilai true dan false saja? Pada pembahasan tipe data sudah pernah disebutkan bahwa boolean merupakan salah satu kunci dari logika pemrograman, karena boolean dapat mengontrol aliran pada program.

Lantas bagaimana cara boolean mengontrol sebuah aliran program? Pada bab selanjutnya, kita akan membahas mengenai if/else statement yang dapat mengontrol flow pada program, tentunya pada penggunaan statement ini boolean sangat berperan.

### If/Else Statement

Ketika mengembangkan sebuah program tentu terdapat alur atau flow proses ketika kode dijalankan. Kita dapat mengontrol alur program ketika suatu kondisi terjadi, misalkan jika nilai x > 5 maka program harus melakukan statement **a**. Jika tidak, program akan menjalankan statement **b**. Untuk melakukan hal tersebut, kita dapat menggunakan if/else statement.

If/else statement dapat digambarkan seolah-olah kita memberikan pertanyaan benar atau salah pada JavaScript, lalu memberikan perintah sesuai jawaban dari pertanyaan tersebut.

Contohnya, terdapat variabel x dengan nilai **50**, kemudian kita bertanya “Hai JavaScript! Apakah x lebih dari 70?” jika kondisi tersebut benar, maka kita dapat memerintahkan JavaScript untuk menampilkan nilainya. Jika salah, kita perintahkan JavaScript untuk menampilkan teks “Nilai kurang dari 70”.

Skenario tersebut dapat dituliskan dalam bentuk kode pada JavaScript seperti berikut:

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5803#run1-editor1)



1

let x = 50;

2

​

3

if(x > 70) {

4

console.log(x);

5

} else {

6

console.log("Nilai kurang dari 70");

7

}

8

​

9

/\* output

10

Nilai kurang dari 70

11

\*/

 Input  Reset

 Jalankan

Nilai kurang dari 70

Pada kode di atas, kita menggunakan kata kunci if untuk memberikan pertanyaan pada JavaScript. Pertanyaan tersebut berupa logical statement yang sudah kita pelajari pada materi sebelumnya. Logical statement pada if ditulis di dalam tanda kurung (parentheses). Jika logical statement tersebut menghasilkan true, maka JavaScript akan mengeksekusi kode yang berada di dalam block if. Jika logical statement menghasilkan nilai false, maka kode yang pada block else lah yang akan dieksekusi.

Block else bersifat opsional untuk dituliskan, hal tersebut berarti kita tidak perlu menuliskannya jika tidak kita manfaatkan. Contohnya:

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5803#run2-editor1)



1

let language = "English";

2

let greeting = "Selamat Pagi!"

3

​

4

if(language === "English") {

5

greeting = "Good Morning!";

6

}

7

​

8

console.log(greeting);

9

​

10

/\* output

11

Good Morning!

12

\*/

 Input  Reset

 Jalankan

Pada kode di atas, nilai greeting memiliki nilai standar “**Selamat Pagi!**” tetapi akan ditampilkan berbeda jika language memiliki nilai “English”, pada kasus ini kita tidak perlu menuliskan block else.

Kita juga dapat memberikan lebih dari satu pertanyaan pada if statement dengan kata lain, kita dapat memberikan lebih dari satu kondisi dengan menggabungkan keyword else dan if seperti berikut:

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5803#run3-editor1)



1

let language = "French";

2

let greeting = "Selamat Pagi"

3

​

4

if(language === "English") {

5

greeting = "Good Morning!";

6

} else if(language === "French") {

7

greeting = "Bonjour!"

8

} else if(language === "Japanese") {

9

greeting = "Ohayogozaimasu!"

10

}

11

​

12

console.log(greeting);

13

​

14

/\* output

15

Bonjour!

16

\*/

 Input  Reset

 Jalankan

Bonjour!

Dengan mengubah nilai dari variabel language ke nilai yang lain seperti **“English”** atau **“Japanese”** maka pesan yang ditampilkan pada console akan ikut berubah.

### Loop

Sejauh ini kita sudah mengenal array sebagai tempat untuk menyimpan banyak data. Ada beberapa kasus dimana kita ingin memeriksa setiap item dalam sebuah array dan melakukan sesuatu dengannya, tetapi kita tidak ingin menuliskan seluruh daftar nilai pada array secara manual, dan menyibukkan diri untuk melakukan hal yang berulang. Maka dari itu kita perlu mempelajari teknik yang dapat mengatasi permasalahan tersebut, teknik ini disebut dengan **loops**.

#### For loop

Terdapat beberapa cara dalam melakukan proses loop pada JavaScript, namun **“for”** merupakan salah satu cara yang banyak digunakan. Struktur dasar dari for tampak seperti berikut:

1. for(inisialisasi variabel; test kondisi; perubahan nilai variabel) {
2. // do something
3. }

Dan berikut contoh penerapan for secara nyata:

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5804?from=5803#run1-editor1)



1

for(let i = 0; i < 5; i++) {

2

console.log(i);

3

}

4

​

5

/\* output

6

0

7

1

8

2

9

3

10

4

11

\*/

 Input  Reset

 Jalankan

0 1 2 3 4

Mungkin kode tersebut sulit dipahami jadi mari kita bahas sedikit demi sedikit.

Yang pertama adalah for(), kita memanggil for statement di mana kita memerintahkan JavaScript seperti ini: “Hai JavaScript! Lakukan perulangan jika kondisi ini benar”, selanjutnya tentu kita membutuhkan sebuah kondisi untuk dievaluasi.

Namun sebelum memberikan kondisi, kita dapat membuat sebuah variabel i sebagai index iterasi dengan memberikan nilai 0 (let i = 0;). Mengapa kita memberikan nilai 0 pada variabel i? Karena ini merupakan sebuah habit dimana perhitungan perulangan dimulai dari 0, begitu juga dengan indexing pada array.

Lalu **i < 5;**merupakan sebuah kondisi dimana jika kondisinya terpenuhi proses looping akan dijalankan. Jadi jika kita memberikan kondisi seperti ini: **i < 5;**maka proses looping akan terjadi sebanyak 5 kali.

Dan terakhir **i++;** menunjukan perubahan nilai variabel i di setiap proses perulangan terjadi. Biasanya perubahan merupakan increment ataupun decrement dari variabel yang kita tetapkan sebelumnya (variabel i). Jika kita tidak menetapkan perubahan nilai, proses perulangan dapat berjalan selamanya! Karena kondisi akan terus terpenuhi.

Lalu { console.log(i); } merupakan statement yang akan dieksekusi pada setiap proses loop. Kita dapat menuliskan banyak statement di sini selama berada di dalam tanda { }. Pada kode tersebut kita memerintahkan JavaScript untuk menampilkan nilai i pada setiap proses perulangan. Sehingga output akan menghasilkan deretan angka dari 0 hingga 4.

Huft, cukup panjang untuk memahami for loops, mari kita kembali ke tujuan awal, lantas bagaimana cara memeriksa item dalam array dengan menggunakan for loop? Kita dapat melakukannya dengan seperti ini:

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5804?from=5803#run2-editor1)



1

const myArray = ["Harry", "Ron", "Hermione", "Tom"];

2

​

3

for(let i = 0; i < myArray.length; i++) {

4

console.log(myArray[i]);

5

}

6

​

7

/\* output

8

Harry

9

Ron

10

Hermione

11

Tom

12

\*/

 Input  Reset

 Jalankan

Harry Ron Hermione Tom

Mudah kan? Kita gunakani < myArray.length sebagai batasan pada proses looping, sehingga proses looping tidak lebih dari panjang array. Kemudian untuk mengakses index array kita manfaatkan nilai variabel i, dimana variabel i ini terus bertambah nilainya pada tiap proses loop, sehingga kita bisa mengakses tiap index pada array secara otomatis.

#### For of loop

Cara lain dalam melakukan looping adalah menggunakan for..of. **For of** mulai hadir pada ECMAScript 2015 (ES6). Cara ini jauh lebih sederhana dan modern dibanding for loop biasa. Struktur dasar dari **for of** loop nampak seperti ini:

1. for(arrayItem of myArray) {
2. *// do something*
3. }

Yups, for of tidak membutuhkan banyak statement untuk melakukan proses looping pada array. Dengan **for..of**, nilai tiap array akan diinisialisasi pada variabel baru yang kita tentukan pada tiap proses looping-nya. Jumlah proses looping nya pun akan menyesuaikan dengan ukuran dari array. Sederhananya seperti kita melakukan perintah “Hei JavaScript! Lakukan perulangan pada myArray, akses tiap nilainya dan simpan pada variabel arrayItem”. Pada proses looping kita gunakan variabel arrayItem untuk mengakses tiap nilai dari item myArray.

Agak sulit memang menjelaskan dengan kata-kata, mari kita terjemahkan dalam kode secara langsung.

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5804?from=5803#run3-editor1)



Lebih mudah dan simpel bukan?

### Function

Tanpa kita sadari sebenarnya kita sudah menggunakan sebuah fungsi pada contoh kode yang ada sebelumnya. console.**log()** (lebih tepatnya pada log()) merupakan sebuah function yang berfungsi untuk menampilkan data pada console browser. Tapi sebenarnya apa itu function? Bagaimana ia bisa bekerja?

Function atau fungsi merupakan potongan kecil kode yang tidak akan dieksekusi sebelum dipanggil. Contoh lain fungsi JavaScript yang sudah ada pada browser adalah alert(). Ia bertujuan untuk menampilkan pesan dalam bentuk pop up dialog. Fungsi tersebut sudah ada pada browser dan akan muncul hanya ketika kita menggunakan/memanggilnya dengan alert().

Atau dalam arti lain, kita dapat berfikir bahwa function merupakan sebuah variabel yang berisi block logika, dan block logika tersebut akan dieksekusi ketika variabelnya dipanggil.

Semua fungsi memiliki struktur yang sama. Nama fungsi selalu diikuti dengan tanda kurung (parentheses) tanpa spasi, lalu terdapat sepasang kurung kurawal yang berisi logika dari fungsi tersebut.

Terkadang di dalam tanda kurung kita membutuhkan sebuah informasi tambahan yang disebut dengan arguments. Argument merupakan data yang digunakan pada fungsi yang dapat mempengaruhi perilaku dari fungsinya tersebut. Contoh, fungsi alert() dapat menerima argument string yang digunakan untuk menampilkan teks pada dialog.

Berikut merupakan ilustrasi dari struktur fungsi.

Terdapat dua tipe fungsi pada JavaScript, yakni native function dan custom function.

Native function merupakan fungsi yang sudah terdapat pada JavaScript atau Browser sehingga kita tidak perlu membuat hanya tinggal menggunakan saja. Contohnya alert(), confirm(), Date(), parseInt() dll. Sebenarnya terdapat ratusan native function yang tersedia.

Custom function merupakan fungsi yang kita buat sendiri, tentu custom function dibuat menyesuaikan kebutuhan kita. Untuk membuat sebuah custom function, kita perlu menuliskan keyword function dilanjutkan dengan menuliskan seluruh struktur fungsinya.

1. function greeting() {
2. console.log("Good Morning!")
3. }

Kemudian kita dapat memanggil fungsinya tersebut dengan menggunakan greeting().

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5809?from=5804#run1-editor1)



1

function greeting(){

2

console.log("Good Morning")

3

}

4

greeting();

 Input  Reset

 Jalankan

Good Morning

Tetapi jika sebuah fungsi hanya menjalankan baris kode secara sama dirasa kurang fungsional bukan? Kita dapat membuat fungsi tersebut untuk menerima argumen dan memanfaatkan argumen untuk mengubah perilaku dari fungsinya.

Untuk menambahkan argument pada fungsi, tambahkan variabel di dalam tanda kurung fungsi namun variabel tersebut tidak memerlukan keyword var, let, ataupun const. Kita juga bisa menambahkan lebih dari satu argumen dengan memberikan tanda koma antar variabel argumennya. Contohnya fungsi greeting akan kita ubah dengan menambahkan argument, sehingga akan nampak seperti ini:

1. function greeting(name, language) {
2. if(language === "English") {
3. console.log("Good Morning " + name + "!");
4. } else if (language === "French") {
5. console.log("Bonjour " + name + "!");
6. } else {
7. console.log("Selamat Pagi " + name + "!");
8. }
9. }

Sehingga dalam memanggilnya pun kita perlu mengirimkan dua buah nilai pada fungsinya seperti berikut:

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5809?from=5804#run2-editor1)



1

​

2

function greeting(name,language){

3

if(language === "french"){

4

console.log("Bonjour!" + name);

5

}else if(language === "japanese"){

6

console.log("Ohayogozaimasu " + name);

7

} else {

8

console.log("Good Morning!" + name);

9

}

10

}

11

greeting("kontol","japanese");

 Input  Reset

 Jalankan

Ohayogozaimasu kontol

Satu hal lagi, function dapat mengembalikan sebuah nilai. Hal ini benar-benar sangat berguna dan membuat kita lebih mudah. Dengan nilai kembalian, kita dapat membuat function yang berfungsi untuk melakukan perhitungan matematika dan hasilnya dapat langsung kita masukkan ke dalam sebuah variabel. Contohnya seperti ini:

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5809?from=5804#run3-editor1)



1

function kali(a,b){

2

return a \* b

3

}

4

​

5

let hasil = kali(8,362)

6

console.log(hasil);

 Input  Reset

 Jalankan

2896

Untuk membuat nilai kembalian dari fungsi gunakan keyword return diikuti dengan nilai yang akan dikembalikan. Nilai kembalian tidak hanya number, bisa saja berupa string, boolean, objek ataupun array. Seperti inilah fungsi greeting() jika kita ubah dengan menetapkan dengan nilai kembalian string:

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5809?from=5804#run4-editor1)



1

function greeting(name, language) {

2

if(language === "English") {

3

return "Good Morning " + name + "!";

4

} else if (language === "French") {

5

return "Bonjour " + name + "!";

6

} else {

7

return "Selamat Pagi " + name + "!";

8

}

9

}

10

​

11

let greetingMessage = greeting("Harry", "French");

12

console.log(greetingMessage);

13

​

14

/\* output

15

Bonjour Harry!

16

\*/

 Input  Reset

 Jalankan

Yang perlu kita perhatikan lagi, ketika statement return tereksekusi, maka fungsi akan langsung terhenti dan mengembalikan nilai.

### Variable Scope

Sejauh ini kita sudah mengenal function. Ada satu hal lagi yang harus kita tahu mengenai dasar JavaScript, yakni scoping variable. Ada banyak keadaan di mana kita membutuhkan variabel untuk diakses di seluruh script yang kita buat. Tetapi ada juga keadaan di mana kita ingin variabel tersebut hanya dapat diakses pada cakupan fungsi dan fungsi turunannya saja.

Variabel yang dapat di akses dari seluruh script disebut dengan “globally scoped,” sementara variabel yang hanya dapat diakses hanya pada function tertentu disebut dengan “locally scoped.”

Variabel JavaScript menggunakan fungsi untuk mengelola cakupannya, Jika variabel didefinisikan di luar fungsi, maka variabel akan bersifat global. Jika variabel didefinisikan di dalam fungsi, maka variabel bersifat lokal dan cakupannya hanya pada fungsi tersebut atau turunannya.

Untuk lebih jelasnya, berikut variabel yang dapat diakses dalam sebuah fungsi:

* Variabel argumen dari fungsinya.
* Lokal variabel yang didefinisikan pada fungsinya.
* Variabel dari induk fungsinya.
* Global variabel.

1. *// global variable, dapat diakses pada parent() dan child()*
2. const a = 'a';
4. function parent() {
5. *// local variable, dapat diakses pada parent() dan child(), tetapi tidak dapat diakses diluar dari fungsi tersebut.*
6. const b = 'b';
8. function child() {
9. *// local variable, dapat diakses hanya pada fungsi child().*
10. const c = 'c';
11. }
12. }

Kita harus berhati-hati dalam mendefinisikan variabel di dalam fungsi. Pasalnya,  kita bisa mendapatkan hasil yang tidak diperkirakan, contohnya seperti berikut:

* [**main.js**](https://www.dicoding.com/academies/123/tutorials/5816?from=5809#run1-editor1)



1

function multiply(num) {

2

total = num \* num;

3

return total;

4

}

5

​

6

let total = 9;

7

let number = multiply(20);

8

​

9

console.log(total)

10

​

11

/\* output

12

400

13

\*/

 Input  Reset

 Hentikan

Mungkin kita berharap nilai total akan tetap 9. Mengingat variabel total pada fungsi multiply, seharusnya tidak akan berpengaruh untuk kode di luar dari fungsi tersebut. Hal ini bisa terjadi karena pada fungsi multiply() kita tidak menetapkan variabel total sebagai cakupan lokal, kita tidak menggunakan keyword const, let, atau var ketika mendeklarasikan variabel total pada fungsi multiply() sehingga variabel total menjadi global.

Perlu kita perhatikan bahwa, ketika kita lupa menuliskan keyword let, const atau var pada script ketika membuat sebuah variabel, maka variabel tersebut akan menjadi global.

Sebisa mungkin kita harus menghindari pembuatan variabel global, karena variabel global dapat diakses pada seluruh script yang kita tuliskan. Semakin banyak variabel global yang kita tuliskan, semakin tinggi kemungkinan variabel tabrakan (collision) terjadi

## Mode Strict pada JavaScript

Seringkali dalam menggunakan JavaScript, kode yang telah ditulis tidak dapat dieksekusi dengan baik pada situs web. Salah satu penyebabnya adalah penulisan yang salah atau menggunakan sintaksis yang buruk.  Untuk menangani hal ini, telah disediakan mode Strict (Strict Mode) sejak versi ECMAScript 5. Hampir semua versi browser terbaru telah mendukung penggunaan mode ini. Internet Explorer mulai versi 10, Firefox mulai versi 4, Google Chrome mulai versi 13, Safari mulai versi 5.1 dan Opera mulai versi 12.

### Apa itu mode Strict

Mode Strict adalah mode yang memaksakan penulisan program JavaScript yang baik dan aman. Hal ini memastikan tidak terjadinya kegagalan dalam menjalankan kode akibat penulisan script yang buruk, dan juga mencegah timbulnya bug.

Berikut beberapa hal yang tidak dapat dilakukan jika menggunakan mode Strict:

1. Menggunakan variabel sebelum mendeklarasikannya.
2. Menghapus variabel, function dan argumen.
3. Menggunakan definisi object property yang berulang.
4. Menggunakan definisi parameter pada function yang berulang.
5. Menulis pada property jenis readonly.
6. Menggunakan angka dengan penulisan oktal.
7. Menulis pada property jenis get.
8. Menghapus undelete property seperti pada property jenis prototype.
9. Menggunakan string “eval” dan “argument” sebagai variabel.
10. Menggunakan statement jenis with.
11. Menggunakan future keyword reserve seperti implements, interface, package, private, protected, public, static, yield.

### Cara aktivasi mode Strict

Jalankan mode Strict dengan cara menuliskan perintah “use strict” pada awal sebuah script atau function. Perintah ditulis dalam bentuk tulisan biasa dan bukan merupakan statemen. Jika perintah ditulis di awal sebuah script maka akan berlaku untuk secara global. Jika ditulis di dalam function maka tidak akan berlaku secara global.

Contoh penggunaan mode Strict secara global (ditulis di awal script)

1. "use strict";
2. x = 9;       // ini akan dianggap error karena variabel belum dideklarasikan
3. showAngka();   // ini akan dianggap error karena function belum dideklarasikan
5. function showAngka() {
6. var x = 9;
7. alert(x);
8. }

Contoh penggunaan mode Strict di dalam function

1. x = 9; // ini tidak dianggap error karena tidak kena mode Strict
2. showAngka();   // ini tidak dianggap error karena tidak kena mode Strict
4. function showAngka() {
5. “use strict”;
6. x = 9; // ini akan dianggap error karena variabel belum dideklarasikan
7. alert(x);
8. }

## Browser Object

Seperti yang kita tahu, dengan menggunakan JavaScript, website kita dapat lebih fungsional dan interaktif. Untuk membuat website menjadi interaktif, tentu JavaScript harus dapat mengontrol elemen yang ada pada website. Dalam mengontrol browser, JavaScript menggunakan object yang sudah tersedia dalam browser yakni window.

Pada JavaScript, browser dikenal sebagai window object. Di dalam objek window itu sendiri terdapat banyak properties dan method yang bisa digunakan, salah satu yang sudah kita ketahui adalah alert().

Kita bisa melihat secara lengkap apa saja properti, method serta event yang ada pada objek window melalui console browser dengan mengetikkan **window** pada console.

Sungguh banyak bukan? Yups dengan properti dan method tersebut, JavaScript mampu mengontrol browser melalui syntax. Kita tidak perlu mengetahui seluruh properti dan method. Berikut beberapa properti dan method yang sering digunakan seperti:

| **Property/Method** | **Description** |
| --- | --- |
| **history** | Sebagai navigasi (go back atau go forward) histori URL browser. |
| **location** | Mendapatkan URL yang terdapat pada address bar browser. |
| **alert()** | Menampilkan dialog alert dengan pesan dan tombol “ok”. |
| **close()** | Menutup tab yang aktif. |
| **confirm()** | Menampilkan dialog dengan pesan dan tombol “ok” dan “cancel”. Method ini akan mengembalikan nilai boolean sesuai response dari pengguna. |
| **prompt()** | Menampilkan dialog dengan pesan dan teks input. Method ini akan mengembalikan nilai string sesuai response dari pengguna. |

Dalam mengakses properti dan method pada objek window, kita bisa menuliskannya secara langsung tanpa harus memanggil objek window-nya terlebih dahulu. Karena properti dan method pada window bersifat global.

1. *// mengakses method alert() pada window*
2. window.alert("Hello Browser!")
4. *// Kita juga bisa mengaksesnya seperti ini*
5. alert("Hello Browser!")

Untuk menggunakan properti dan method yang terdapat pada objek window, tentu syntax JavaScript perlu dijalankan pada browser atau dituliskan pada console browser. Kita tidak bisa mencobanya pada [glot.io](https://glot.io/) seperti saat belajar sintaks dasar JavaScript, karena glot.io sejatinya tidak memiliki objek window dan proses kompilasi JavaScript dilakukan menggunakan teknologi [nodeJS](https://nodejs.org/en/). Sehingga jika kita mencoba menjalankan syntax JavaScript yang berhubungan dengan browser objek akan menyebabkan eror “***not defined***”.

Berbeda ketika kita menjalankan syntax tersebut pada console browser. Method alert() akan berhasil dijalankan dan memunculkan pesan dialog.

Setelah mengetahui cara penggunaan browser object dan dasar-dasar JavaScript, dengan menggabungkan pengetahuan keduanya, kita bisa membuat program sederhana. Contohnya kita bisa membuat program yang interaktif dengan menggunakan prompt() untuk mendapatkan nilai input dari pengguna dan alert() untuk menampilkan pesan.

Silakan tuliskan kode berikut ini dan jalankan pada console browser.

1. const firstName = prompt("Siapa nama depanmu?");
2. const lastName = prompt("Siapa nama belakangmu?");
3. const language = prompt("Bisa berbahasa apa?");
5. const user = {
6. name: {
7. first: firstName,
8. last: lastName,
9. },
10. language: language
11. };
13. if (user.language === "English") {
14. alert("Nice to meet you " + user.name.first + " " + user.name.last + "!");
15. } else if (user.language === "French") {
16. alert("Ravi de vous rencontrer " + user.name.first + " " + user.name.last + "!");
17. } else if (user.language === "Japanese") {
18. alert("Hajimemashite, " + user.name.first + " " + user.name.last + "!");
19. } else {
20. alert("Senang bertemu dengan Anda " + user.name.first + " " + user.name.last + "!");
21. }

Maka hasilnya akan tampak seperti ini:

## Manipulasi DOM

Sekarang kita sudah memahami bahasa JavaScript dan mengerti peran penggunaannya dalam browser melalui objek window. Tetapi ada satu hal lagi yang belum kita ketahui, yakni cara mengakses elemen pada dokumen HTML.

Pada materi kali ini kita akan eksplorasi tentang DOM Scripting, di mana nantinya kita dapat memanipulasi elemen, atribut elemen, dan teks yang tampak pada halaman.

### Apa itu DOM?

Sebelumnya disebutkan DOM Scripting, tapi sebenarnya apa itu DOM? Document Object Model (DOM) memberikan kita jalan untuk mengakses dan memanipulasi konten pada dokumen. DOM merupakan application programming interface (API) untuk HTML, XML atau SVG. Dengan DOM berkas HTML dapat direpresentasikan dalam bentuk objek yang dapat diakses oleh JavaScript (sebenarnya tidak hanya oleh JavaScript. DOM juga dapat diakses oleh bahasa pemrograman lain). Melalui DOM inilah JavaScript dapat memanipulasi elemen beserta atributnya pada HTML.

Struktur objek DOM digambarkan seperti struktur node tree (pohon simpul). Disebut pohon karena strukturnya seperti pohon dengan batang induk tunggal yang bercabang menjadi beberapa cabang anak, masing-masing memiliki daun. Pada kasus ini induk tunggal batang merupakan elemen <html>, cabangnya merupakan anak elemen di dalamnya, dan daun adalah konten di dalam elemen tersebut.

Mari kita ambil contoh pada struktur HTML berikut:

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <title>DOM Structure</title>
5. </head>
6. <body>
7. <h1>Hallo Developer!</h1>
8. <p>Belajar Dasar Pemrograman Web</p>
9. </body>
10. </html>

Berdasarkan HTML di atas, maka objek DOM memiliki struktur seperti ini:

Pada contoh yang diberikan di atas DOM terlihat seperti pemetaan dari sumber berkas HTML. Walaupun sama, berkas HTML dan DOM adalah hal yang berbeda.

Walaupun struktur dari DOM terbentuk berdasarkan berkas HTML, tetapi terdapat kasus tertentu yang menyebabkan struktur DOM berbeda dengan struktur HTML. Yakni kasus di mana kita terdapat kesalahan dalam menuliskan kode HTML.

Mari kita lihat contohnya seperti berikut:

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

Pada berkas HTML di atas tidak terdapat elemen <head> dan <body>. Artinya berkas HTML tersebut tidak valid. Berbeda dengan DOM, walaupun struktur pada HTML tidak valid, pada DOM object structure akan diperbaiki. Caranya dengan menambahkan elemen <head> dan <body> secara otomatis kemudian menempatkan teks di dalam elemen <body>.

### Mendapatkan Elemen

Untuk mengakses elemen melalui DOM, kita gunakan properti dari objek window yang dinamakan dengan **document**. Objek document merepresentasikan dokumen dari HTML atau keseluruhan halaman yang terlihat pada jendela browser. Jika kita lihat nilai dari objek document pada browser console akan tampak seperti ini:

Pada gambar di atas kita bisa lihat bahwa nilai dari document merupakan struktur HTML dari halaman yang ditampilkan, di mana terdapat root element <html> yang di dalamnya ada elemen <head> dan <body>.

Meskipun nilai objek document terlihat hanya seperti struktur HTML saja, namun objek document memiliki beberapa standar properti dan method yang digunakan untuk mengakses koleksi elemen di dalamnya. Contohnya kita dapat mendapatkan elemen <head> yang diakses melalui properti head dari objek document.

1. const head = document.head;

Begitu juga sama halnya dengan mendapatkan elemen <body> dengan properti body seperti berikut.

1. const body = document.body;

Kedua properties tersebut baik head atau body akan mengembalikan elemen dalam bentuk objek yang sama seperti document. Namun bila kita ingin mengakses elemen secara spesifik, kita harus menggunakan method yang disediakan pada objek document seperti:

| **Method** | **Contoh** | **Fungsi** |
| --- | --- | --- |
| getElementById() | document.getElementById(“display”); | Mengembalikan elemen yang memiliki nilai id “display”. |
| getElementsByName() | document.getElementsByName(“button”) | Mengembalikan banyak elemen (NodeList) yang memiliki attribute name dengan nilai “button”. |
| getElementsByClassName() | document.getElementsByClassName(“button”) | Mengembalikan kumpulan yang memiliki class "button" dalam bentuk HTMLCollection. |
| getElementsByTagName() | document.getElementsByTagName(“div”) | Mengembalikan banyak <div> elemen berupa HTMLCollection |
| querySelector() | document.querySelector(“.button”); | Mengembalikan elemen pertama yang menerapkan class “button”. |
| querySelectorAll() | document.querySelectorAll(“.button”); | Mengembalikan kumpulan Node beserta turunannya (NodeList) dengan class “button”. |

Pada method di atas ada yang mengembalikan nilai HTML elemen secara tunggal, ada juga yang mengembalikan banyak nilai HTML elemen yang biasa disebut HTMLCollection. Contohnya kita gunakan method document.querySelector(".button"); untuk mendapatkan nilai button. Jadi, nilai yang didapatkan adalah elemen tunggal yang menerapkan class “button” pertama kali.

Untuk mendapatkan seluruh elemen yang memiliki class “button” kita gunakan method document.querySelectorAll(".button"); sehingga method tersebut akan mengembalikan banyak elemen dengan class button.

Karena HTMLCollection mirip seperti array yang dapat menampung banyak data di dalamnya, HTMLCollection juga memiliki karakteristik mirip seperti array. HTMLCollection memiliki properti length untuk mendapatkan jumlah elemen yang ditampung. Untuk mengakses nilai individual elemennya menggunakan indexing.

Kita juga dapat melakukan perulangan menggunakan for of pada HTMLCollection:

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

Dalam menggunakan method querySelector() tentu kita menggunakan query dalam menentukan target elemen. Untuk menargetkan elemen berdasarkan attribute class, kita gunakan tanda “.”, sedangkan jika kita menargetkan dengan menggunakan id, gunakan tanda “#”.

Selain querySelector() masih terdapat method lain yang sudah disebutkan sebelumnya untuk mendapatkan elemen secara spesifik, Anda bisa mencoba sendiri method tersebut pada console browser, dan carilah perbedaan fungsi dari masing-masing method-nya

### Memanipulasi Element

Setelah kita tahu cara mengakses elemen melalui objek document, sekarang saatnya kita belajar cara memanipulasi elemen yang didapat, seperti mengubah konten, menambahkan atau mengubah nilai atribut, dan menambahkan action/event pada elemen.

#### Memanipulasi Atribut

Untuk mengubah atau menambah nilai atribut pada elemen, apa  method yang kita gunakan? setAttribute() jawabannya. Method ini membutuhkan dua buah argumen string yang merupakan nama dan nilai dari atributnya.

1. someElement.setAttribute("attributName", "attributeValue");

Mari kita ambil contoh. Terdapat struktur web sebagai berikut:

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <title>Manipulating Attribute</title>
5. </head>
6. <body>
7. <figure>
8. <img id="catImage" src="https://i.ibb.co/9WT6LG6/two-three.jpg" alt="three-cats" width="600px"/>
9. <figcaption id="caption">Tiga Anak Kucing</figcaption>
10. </figure>
11. </body>
12. </html>

Kita buka berkas HTML tersebut pada browser, maka akan muncul tampilan seperti ini:

Ada yang janggal pada informasi yang ditampilkan di website. Berdasarkan caption, gambar seharusnya menampilkan 3 ekor kucing. Kita akan coba memperbaikinya dengan mengubah nilai atribut src pada elemen <img id=”catImage”> melalui JavaScript. Kita bisa melakukannya dengan menuliskan kode seperti berikut:

1. let catImage = document.querySelector("#catImage");
2. catImage.setAttribute("src", "https://i.ibb.co/55VG7vL/three-cat.jpg");

Jalankan kode berikut pada console browser kemudian gambar pada halaman akan berubah.

Selain mengatur nilai atribut elemen, kita juga dapat mengambil nilai dari atribut elemen yang sedang diterapkan dengan menggunakan method getAttribute(). Method tersebut membutuhkan satu buah argumen string yang merupakan nama atribut dan akan mengembalikan nilai dari atribut tersebut. Contoh:

1. catImage.getAttribute("src");

#### Memanipulasi Konten Elemen

Dengan JavaScript juga kita dapat mengubah konten yang ada di dalam HTML. Konten pada elemen disimpan di dalam properti innerHTML (konten dalam bentuk HTML) atau innerText (konten dalam bentuk Teks). Hasilnya, dalam memanipulasi konten kita dapat melakukannya dengan mengubah nilai dari properti tersebut.

1. let caption = document.querySelector("#caption");
2. caption.innerHTML = '<em>Tiga Anak Kucing</em>'

Lihat perubahan terjadi pada caption gambar! Sekarang caption gambar tampak miring karena kita membungkus teks dalam elemen emphasize. Kita dapat memberikan nilai HTML pada konten elemen dengan menggunakan properti innerHTML.

Berbeda ketika kita menggunakan innerText, nilai apapun yang terdapat pada properti ini adalah merupakan sebuah teks. Sehingga walaupun kita menginisialisasi teks yang merupakan sebuah tag html, maka teks tersebut akan tampil apa adanya tanpa terformat sebagai HTML.

1. caption.innerText = "<em>Tiga Anak Kucing</em>";

Jadi pastikan kita tidak salah menggunakan properti ketika ingin mengubah konten dari elemen.

#### Membuat dan Menambahkan Elemen Baru

Sebenarnya terdapat beberapa cara dalam membuat elemen HTML baru menggunakan JavaScript. Bahkan kita bisa gunakan langsung properti innerHTML dengan menuliskan langsung tag HTML-nya kemudian menggabungkan dengan nilai yang ada pada elemen (appending). Tapi ini bukan pendekatan yang baik.

Dalam membuat elemen baru, DOM telah menyediakan method yang bernama createElement().

1. let newElement = document.createElement('p');

Dengan menjalankan perintah tersebut maka terciptalah elemen paragraf baru yang diinisialisasi pada variabel newElement.

Pada variabel newElement kita dapat memberikan nilai konten dengan menggunakan properti innerHTML ataupun innerText.

1. newElement.innerHTML = 'Anda menekan gambar kucing sebanyak <span id="count">0</span> kali';

Walaupun sekarang variabel newElement sudah memiliki konten di dalamnya, tetapi mengapa belum muncul pada jendela browser? Hal tersebut karena fungsi createElement() hanya akan membuat sebuah elemen baru, tidak berarti ia akan memasukkannya ke dalam document.body.

Agar elemen baru tampil pada jendela browser, kita perlu memasukkan elemen tersebut pada body dengan menggunakan fungsi appendChild();

1. document.body.appendChild(newElement);

Sekarang elemen baru yang kita buat sudah tampil pada jendela browser.

#### Menambahkan Aksi (Event) pada Element

Selain mampu mengakses elemen dalam bentuk objek pada halaman, JavaScript juga mampu menerima event atau kejadian yang terjadi pada elemen. Hal tersebut dinamakan Event Handler. Event bisa berupa interaksi dari pengguna seperti click, atau sekedar mengarahkan kursor pada elemen. Dengan menerapkan event handler, kita dapat menjalankan suatu fungsi tertentu ketika event terjadi pada elemen.

Untuk menambahkan Event Handler pada elemen kita gunakan method, addEventListener() pada target element. Method ini membutuhkan setidaknya dua buah argument.

Yang pertama adalah sebuah string sebagai tipe event-nya. Ada banyak sekali tipe event yang dapat digunakan pada method addEventListener(), Anda bisa melihat secara lengkap pada tautan berikut: <https://developer.mozilla.org/en-US/docs/Web/Events>. Salah satu yang banyak digunakan adalah “**click**”. Event ini akan membaca kejadian dimana pengguna melakukan click pada element.

Kemudian yang kedua adalah sebuah fungsi yang akan dijalankan ketika event terjadi. Pada fungsi ini kita dapat memberikan sebuah argumen yang merupakan object yang berisikan informasi tentang action yang terjadi. Termasuk informasi mengenai target elemen event-nya (event.target). Argumen ini biasanya diberikan nama **“event”**.

Lihat kode berikut untuk menerapkan addEventListener() pada sebuah elemen.

1. catImage.addEventListener('click', function(event) {
2. document.querySelector('#count').innerText++;
3. });

Dengan menjalankan kode tersebut pada latihan sebelumnya, berarti kita memberikan sebuah event ‘click’ pada elemen catImage. Ketika gambar kucing ditekan akan menjalankan fungsi yang menambahkan nilai konten pada elemen dengan id count.

Seperti itulah dasar-dasar dalam memanipulasi DOM dengan menggunakan JavaScript. Untuk mengimplementasikan apa yang sudah kita pelajari, pada materi selanjutnya kita akan coba terapkan logika pada Web Kalkulator agar dapat berfungsi dengan baik.

**Menerapkan JavaScript pada Web Kalkulator**

Sejauh ini kita sudah mempelajari bagaimana cara menggunakan JavaScript pada HTML, bagaimana menuliskan sintaks JavaScript, mengenal tipe data dan fungsi sintaks yang ada, hingga akhirnya memanipulasi dan memberikan event pada elemen HTML melalui DOM *Object*.

Dengan komponen yang sudah disebutkan tadi, sepertinya sekarang kita sudah cukup bekal untuk memberikan “nyawa” pada Web Kalkulator kita sehingga dapat berfungsi layaknya sebuah kalkulator pada umumnya. Let’s do it!

Silakan buka project Web Kalkulator dengan editor favorit kita.

Jika Anda mengikuti seluruh latihan sebelumnya, maka struktur projek tampak seperti gambar di atas. Kita juga sudah mencoba menghubungkan berkas JavaScript (**kalkulator.js**) dengan berkas HTML (**index.html**) lalu menuliskan sintaks dasar untuk menampilkan pesan pada console browser.

Selanjutnya kita akan bekerja full pada **kalkulator.js**. Silakan hapus sintaks yang sudah kita buat sebelumnya.

1. console.log('Selamat Anda berhasil menggunakan JavaScript pada Website');

Langkah pertama adalah buatlah sebuah objek dengan nama calculator. Di dalamnya terdapat properti yang menggambarkan data dan kondisi dari kalkulatornya, seperti displayNumber, operator, firstNumber, dan waitingForSecondNumber. Sehingga kodenya akan nampak seperti ini:

1. const calculator = {
2. displayNumber: '0',
3. operator: null,
4. firstNumber: null,
5. waitingForSecondNumber: false,
6. };

Kita gunakan objek ini sebagai tempat menyimpan data dan kondisi pada calculator, di mana angka yang muncul pada layar kalkulator selalu diambil dari data calculator.displayNumber.

Properti operator dan firstNumber kita gunakan nilai null terlebih dahulu karena properti tersebut akan diberikan nilai ketika pengguna melakukan aksi.

Dan properti waitingForSecondNumber merupakan kondisi di mana kalkulator sedang menunggu pengguna menentukkan angka kedua dalam melakukan perhitungan.

Setelah membuat *object* calculator, selanjutnya kita buat fungsi-fungsi umum yang dilakukan kalkulator seperti me-*update* angka pada layar dan menghapus data pada kalkulator.

1. function updateDisplay() {
2. document.querySelector('#displayNumber').innerText = calculator.displayNumber;
3. }
5. function clearCalculator() {
6. calculator.displayNumber = '0';
7. calculator.operator = null;
8. calculator.firstNumber = null;
9. calculator.waitingForSecondNumber = false;
10. }

Lalu kita buat juga fungsi untuk memasukkan angka ke dalam nilai displayNumber kalkulator.

1. function inputDigit(digit) {
2. calculator.displayNumber += digit;
3. }

Kemudian kita buat variabel buttons dengan menginisialisasikan nilai seluruh elemen button yang ada, dan berikan event **click** pada tiap elemennya.

Untuk mendapatkan nilai seluruh elemen button kita gunakan querySelectorAll(“.button”) kemudian kita *looping* nilainya dan berikan event **click** pada tiap itemnya.

1. const buttons = document.querySelectorAll('.button');
3. for (const button of buttons) {
4. button.addEventListener('click', function (event) {
5. // mendapatkan objek elemen yang diklik
6. const target = event.target;
8. inputDigit(target.innerText);
9. updateDisplay();
10. });
11. }

Sehingga keseluruhan kode pada **kalkulator.js** akan tampak seperti ini:

1. const calculator = {
2. displayNumber: '0',
3. operator: null,
4. firstNumber: null,
5. waitingForSecondNumber: false,
6. };
8. function updateDisplay() {
9. document.querySelector('#displayNumber').innerText = calculator.displayNumber;
10. }
12. function clearCalculator() {
13. calculator.displayNumber = '0';
14. calculator.operator = null;
15. calculator.firstNumber = null;
16. calculator.waitingForSecondNumber = false;
17. }
19. function inputDigit(digit) {
20. calculator.displayNumber += digit;
21. }
23. const buttons = document.querySelectorAll('.button');
24. for (const button of buttons) {
25. button.addEventListener('click', function (event) {
26. // mendapatkan objek elemen yang diklik
27. const target = event.target;
29. inputDigit(target.innerText);
30. updateDisplay();
31. });
32. }

Sekarang kita coba jalankan **index.html** pada browser, dan lihat fungsi kalkulator untuk yang pertama kali.

Uh, sangat aneh, bukan? Siapa yang ingin gunakan kalkulator seperti itu? Untuk saat ini tak apa. Setidaknya kode yang kita tuliskan sudah berhasil berjalan dengan baik. Selanjutnya kita akan memperbaiki keanehan-keanehan yang ada satu per satu.

Saat ini kalkulator masih dapat menampilkan angka 0 di awal bilangan, hal itu tentu aneh dan tidak pernah terjadi pada kalkulator manapun terkecuali dalam menampilkan bilangan desimal. Untuk memperbaikinya, pada fungsi inputDigit(), tambahkan sebuah kondisi dimana jika displayNumber bernilai ‘0’, maka angka yang pertama dimasukkan pengguna akan menggantikan keseluruhan nilai displayNumber selain itu, lakukan seperti biasanya. Untuk melakukannya kita gunakan *if-else statement*.

1. function inputDigit(digit) {
2. if (calculator.displayNumber === '0') {
3. calculator.displayNumber = digit;
4. } else {
5. calculator.displayNumber += digit;
6. }
7. }

Dengan begitu kalkulator tidak akan menampilkan angka 0 diawal bilangan lagi.

Kemudian kita akan membuat fungsi clear pada kalkulator berjalan dengan semestinya sehingga dalam mereset kalkulator kita tidak perlu melakukan reload pada browser seperti yang ditunjukkan pada GIF di atas.

Pada *event handler*, kita tambahkan kondisi dimana ketika *event target* merupakan elemen yang menerapkan class clear maka kita akan panggil fungsi clearCalculator().

1. button.addEventListener('click', function(event) {
2. // mendapatkan objek elemen yang diklik
3. const target = event.target;
5. if (target.classList.contains('clear')) {
6. clearCalculator();
7. updateDisplay();
8. return;
9. }
11. inputDigit(target.innerText);
12. updateDisplay();
13. });

Kita bisa memanfaatkan event.classList untuk melihat nilai class apa saja dalam bentuk array yang ada pada element target, kemudian menggunakan contains() yang merupakan *method* dari array yang berguna untuk memastikan nilai yang terkandung di dalam array tersebut.

Jika kondisi if terpenuhi sudah dipastikan tombol tersebut adalah tombol clear sehingga kita perlu memanggil fungsi clearCalculator() dan mengupdate *display* kalkulator. Jangan lupa juga untuk gunakan *return statement* agar fungsi *event handler* terhenti sehingga kode yang ada di bawahnya tidak ikut tereksekusi.

Setelah menerapkan kondisi tersebut maka seluruh kode pada **kalkulator.js** akan tampak seperti berikut:

1. const calculator = {
2. displayNumber: '0',
3. operator: null,
4. firstNumber: null,
5. waitingForSecondNumber: false,
6. };
8. function updateDisplay() {
9. document.querySelector('#displayNumber').innerText = calculator.displayNumber;
10. }
12. function clearCalculator() {
13. calculator.displayNumber = '0';
14. calculator.operator = null;
15. calculator.firstNumber = null;
16. calculator.waitingForSecondNumber = false;
17. }
19. function inputDigit(digit) {
20. if (calculator.displayNumber === '0') {
21. calculator.displayNumber = digit;
22. } else {
23. calculator.displayNumber += digit;
24. }
25. }
27. const buttons = document.querySelectorAll('.button');
28. for (const button of buttons) {
29. button.addEventListener('click', function(event) {
30. *// mendapatkan objek elemen yang diklik*
31. const target = event.target;
33. if (target.classList.contains('clear')) {
34. clearCalculator();
35. updateDisplay();
36. return;
37. }
39. inputDigit(target.innerText);
40. updateDisplay();
41. });
42. }

Selanjutnya kita akan melengkapi beberapa fungsi lainnya yang ada pada kalkulator, yakni fungsi *negative*, *operator*, dan *equals*. Untuk itu, buat kondisi lainnya pada *event handler*, sehingga kode tampak seperti berikut.

1. button.addEventListener('click', function (event) {
2. // mendapatkan objek elemen yang diklik
3. const target = event.target;
5. if (target.classList.contains('clear')) {
6. clearCalculator();
7. updateDisplay();
8. return;
9. }
11. if (target.classList.contains('negative')) {
12. inverseNumber();
13. updateDisplay();
14. return;
15. }
17. if (target.classList.contains('equals')) {
18. performCalculation();
19. updateDisplay();
20. return;
21. }
23. if (target.classList.contains('operator')) {
24. handleOperator(target.innerText);
25. return;
26. }
28. inputDigit(target.innerText);
29. updateDisplay()
30. });

Jika kita membukanya sekarang, error akan muncul ketika tombol-tombol fungsi dan operatornya ditekan.

Hal tersebut wajar karena kita belum mendefinisikan seluruh fungsi-fungsi yang kita dituliskan di atas. Dengan begitu mari kita buat fungsinya mulai dari inverseNumber().

1. function inverseNumber() {
2. if (calculator.displayNumber === '0') {
3. return;
4. }
5. calculator.displayNumber = calculator.displayNumber \* -1;
6. }

Fungsi inverseNumber() cukuplah simple karena kita hanya perlu melakukan perkalian displayNumber dengan -1. Terkecuali jika displayNumber masih bernilai ‘0’, perkalian tidak akan dilakukan.

Sekarang tombol **"+/-"** sudah berfungsi dengan baik, selanjutnya kita akan membuat fungsi untuk menetapkan sebuah operator, baik itu + atau - pada kalkulator. Tuliskan fungsi berikut.

1. function handleOperator(operator) {
2. if (!calculator.waitingForSecondNumber) {
3. calculator.operator = operator;
4. calculator.waitingForSecondNumber = true;
5. calculator.firstNumber = calculator.displayNumber;
7. // mengatur ulang nilai display number supaya tombol selanjutnya dimulai dari angka pertama lagi
8. calculator.displayNumber = '0';
9. } else {
10. alert('Operator sudah ditetapkan');
11. }
12. }

Fungsi tersebut membutuhkan satu buah argument yang merupakan sebuah operator. Nilai operator tersebut bersumber dari innerText tombol operator yang menjadi *event target*. Secara prinsip fungsi ini bertujuan untuk menyimpan operator dan firstNumber dengan nilai displayNumber saat ini pada *object* calculator, hanya jika properti waitingForSecondNumber bernilai false. Namun jika waitingForSecondNumber bernilai true, browser akan menampilkan alert() dengan pesan **“Operator sudah ditetapkan”**

Voila! Sekarang tombol operator sudah dapat menetapkan nilai operator pada *object calculator*.

Kita bisa lihat pada console bahwa sekarang nilai properti operator dan firstNumber tidak lagi null, begitu pula dengan properti waitingForSecondNumber yang sudah berubah menjadi true.

Kita buat fungsi terakhir yakni performCalculation(). Fungsi ini digunakan untuk melakukan kalkulasi terhadap nilai - nilai yang terdapat pada objek calculator, sehingga pastikan kalkulator sudah memiliki nilai operator dan firstNumber ketika fungsi ini dijalankan.

1. function performCalculation() {
2. if (calculator.firstNumber == null || calculator.operator == null) {
3. alert('Anda belum menetapkan operator');
4. return;
5. }
7. let result = 0;
8. if (calculator.operator === '+') {
9. result = parseInt(calculator.firstNumber) + parseInt(calculator.displayNumber);
10. } else {
11. result = parseInt(calculator.firstNumber) - parseInt(calculator.displayNumber);
12. }
14. calculator.displayNumber = result;
15. }

Fungsi tersebut diawali dengan pengecekan nilai-nilai yang dibutuhkan untuk melakukan kalkulasi. Jika tidak terpenuhi maka proses akan dihentikan. Namun jika terpenuhi kalkulasi akan dilakukan.

Dalam melakukan kalkulasi terdapat pengecekan tipe operator apa yang akan dilakukan. Kita juga menggunakan parseInt() untuk mengubah nilai **string** menjadi **number**. Mengapa konversi tipe data dibutuhkan? Sejatinya kita menggunakan string dalam menampilkan nilai pada jendela browser, namun untuk proses kalkulasi kita membutuhkan number.

Setelah menambahkan fungsi tersebut, maka seluruh struktur kode pada berkas **kalkulator.js** akan tampak seperti berikut.

1. const calculator = {
2. displayNumber: '0',
3. operator: null,
4. firstNumber: null,
5. waitingForSecondNumber: false,
6. };
8. function updateDisplay() {
9. document.querySelector('#displayNumber').innerText = calculator.displayNumber;
10. }
12. function clearCalculator() {
13. calculator.displayNumber = '0';
14. calculator.operator = null;
15. calculator.firstNumber = null;
16. calculator.waitingForSecondNumber = false;
17. }
19. function inputDigit(digit) {
20. if (calculator.displayNumber === '0') {
21. calculator.displayNumber = digit;
22. } else {
23. calculator.displayNumber += digit;
24. }
25. }
27. function inverseNumber() {
28. if (calculator.displayNumber === '0') {
29. return;
30. }
31. calculator.displayNumber = calculator.displayNumber \* -1;
32. }
34. function handleOperator(operator) {
35. if (!calculator.waitingForSecondNumber) {
36. calculator.operator = operator;
37. calculator.waitingForSecondNumber = true;
38. calculator.firstNumber = calculator.displayNumber;
39. calculator.displayNumber = '0';
40. } else {
41. alert('Operator sudah ditetapkan');
42. }
43. }
45. function performCalculation() {
46. if (calculator.firstNumber == null || calculator.operator == null) {
47. alert('Anda belum menetapkan operator');
48. return;
49. }
51. let result = 0;
52. if (calculator.operator === '+') {
53. result = parseInt(calculator.firstNumber) + parseInt(calculator.displayNumber);
54. } else {
55. result = parseInt(calculator.firstNumber) - parseInt(calculator.displayNumber);
56. }
58. calculator.displayNumber = result;
59. }

62. const buttons = document.querySelectorAll('.button');
63. for (const button of buttons) {
64. button.addEventListener('click', function (event) {
65. // mendapatkan objek elemen yang diklik
66. const target = event.target;
68. if (target.classList.contains('clear')) {
69. clearCalculator();
70. updateDisplay();
71. return;
72. }
74. if (target.classList.contains('negative')) {
75. inverseNumber();
76. updateDisplay();
77. return;
78. }
80. if (target.classList.contains('equals')) {
81. performCalculation();
82. updateDisplay();
83. return;
84. }
86. if (target.classList.contains('operator')) {
87. handleOperator(target.innerText)
88. return;
89. }
91. inputDigit(target.innerText);
92. updateDisplay();
93. });
94. }

Jika kita buka **index.html** sekarang, seharusnya kalkulator sudah dapat melakukan kalkulasi.

Selamat! Sejauh ini kita sudah bisa membuat aplikasi web dengan memanfaatkan pengetahuan yang sudah kita pelajari. Selanjutnya kita akan mengenal salah satu Web API yang menarik untuk diterapkan pada project kalkulator kita.

# Perkenalan ke Web Storage

Kita sudah sampai di penghujung materi dari kelas ini. Selamat bagi Anda yang sudah menguasai dasar dari tiga pilar pembentuk website. Pada materi kali ini  kita akan berkenalan mengenai salah satu web API yang menarik untuk diterapkan pada proyek kalkulator, yakni Web Storage.

## Apa itu Web Storage?

Web storage adalah salah satu Web API yang dapat menyimpan data secara lokal pada sisi client. Berbeda dengan objek atau array, data yang disimpan pada objek atau array JavaScript bersifat sementara, dan akan hilang jika terjadi reload atau pergantian URL pada browser. Sedangkan data yang disimpan pada Web Storage akan bertahan lebih lama karena data akan disimpan pada storage browser.

Sebelum HTML5, kita melakukan hal ini dengan menggunakan [cookies](https://id.wikipedia.org/wiki/Kuki_HTTP), namun penyimpanan cookies terbatas. Dengan hadirnya Web Storage kita dapat menampung data lebih besar dan tentunya lebih aman.

Data yang disimpan web storage tersedia berdasarkan domain. Artinya, data pada suatu domain web hanya dapat diakses oleh domainnya itu sendiri. Web storage dapat menampung sebesar 10MB untuk satu domain.

## Tipe Web Storage

Web API menyediakan dua tipe Web Storage untuk kita gunakan, yakni sessionStorage dan localStorage.

Sebelum menggunakan Web Storage baik sessionStorage atau localStorage, kita perlu memastikan browser support terhadap fitur tersebut, dengan cara:

1. if (typeof(Storage) !== 'undefined') {
2. // Browser mendukung sessionStorage/localStorage.
3. } else {
4. // Browser tidak mendukung sessionStorage/LocalStorage
5. }

Seharusnya browser yang ada pada saat ini sudah mendukung kedua fitur tersebut. Google Chrome dan Mozilla Firefox tentu memilikinya.

Pada sessionStorage atau localStorage data yang disimpan adalah nilai primitif seperti **number**, **boolean**, atau **string**. Namun kita juga dapat menyimpan data berbentuk objek dengan mengubahnya dalam bentuk **string**, begitu pula sebaliknya ketika kita ingin menggunakan datanya kembali.

Untuk menyimpan dan mengakses data pada storage, metode yang digunakan adalah key-value. Di sini key menjadi sebuah kunci untuk mengakses data pada storage.

Data yang disimpan pada Web Storage dapat kita lihat menggunakan DevTools pada tab **Application** (Google Chrome) atau tab **Storage** (Mozilla Firefox).

Data Local Storage pada Google Chrome

Data Local Storage pada Mozilla Firefox

### Session Storage

Tipe storage yang pertama adalah Session Storage yang mana digunakan untuk menyimpan data sementara pada browser. Data akan hilang ketika browser atau tab browser ditutup.

Untuk menggunakan Session Storage kita gunakan object sessionStorage, kemudian untuk menyimpan datanya gunakan method setItem(), method ini membutuhkan dua argumen yakni key dan nilai yang akan dimasukkan. Kemudian untuk mengakses data yang sudah dimasukkan kita gunakan method getItem() dan gunakan key sebagai argumen methodnya.

Berikut contoh penerapan dari session storage:

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <title>Session Storage</title>
5. </head>
6. <body>
7. <p>Anda sudah menekan tombol sebanyak <span id="count"></span> kali</p>
8. <button id="button">Click Disini!</button>
10. <script>
11. const cacheKey = 'NUMBER';
12. if (typeof (Storage) !== 'undefined') {
13. // pengecekkan apakah data sessionStorage dengan key NUMBER tersedia atau belum
14. if (sessionStorage.getItem(cacheKey) === 'undefined') {
15. // Jika belum maka akan atur dengan nilai awal yakni 0
16. sessionStorage.setItem(cacheKey, 0);
17. }
19. const button = document.querySelector('#button');
20. const count = document.querySelector('#count');
21. button.addEventListener('click', function (event) {
22. let number = sessionStorage.getItem(cacheKey);
23. number++;
24. sessionStorage.setItem(cacheKey, number);
25. count.innerText = sessionStorage.getItem(cacheKey);
26. });
27. } else {
28. alert('Browser tidak mendukung Web Storage');
29. }
30. </script>
31. </body>
32. </html>

Bisa kita lihat bahwa data yang disimpan pada sessionStorage akan bertahan jika terjadi reload pada browser, namun data tersebut akan hilang apabila tab browser atau browser itu sendiri ditutup.

### Local Storage

Tipe storage yang kedua adalah Local Storage yang serupa dengan session storage, namun data yang disimpan tidak akan hilang bila browser atau tabs browser ditutup. Kita bisa menghapus data pada local storage dengan method removeItem().

Untuk penggunaan localStorage identik dengan sessionStorage, perbedaannya storage ini diakses melalui object localStorage. Berikut contoh penerapan dari local storage:

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <title>Local Storage</title>
5. </head>
6. <body>
7. <p>Anda sudah menekan tombol sebanyak <span id="count"></span> kali</p>
8. <button id="button">Click Disini!</button>
9. <button id="clear">Hapus Storage</button>
11. <script>
12. const cacheKey = 'NUMBER';
13. if (typeof (Storage) !== 'undefined') {
14. // pengecekkan apakah data localStorage dengan key NUMBER tersedia atau belum
15. if (localStorage.getItem(cacheKey) === 'undefined') {
16. // Jika belum maka akan atur dengan nilai awal yakni 0
17. localStorage.setItem(cacheKey, 0);
18. }
20. const button = document.querySelector('#button');
21. const clearButton = document.querySelector('#clear');
22. const count = document.querySelector('#count');
23. button.addEventListener('click', function (event) {
24. let number = localStorage.getItem(cacheKey);
25. number++;
26. localStorage.setItem(cacheKey, number);
27. count.innerText = localStorage.getItem(cacheKey);
28. });
30. clearButton.addEventListener('click', function (event) {
31. localStorage.removeItem(cacheKey);
32. });
33. } else {
34. alert('Browser tidak mendukung Web Storage');
35. }
36. </script>
37. </body>
38. </html>

Pada gif di atas kita bisa mengetahui bahwa data yang disimpan pada localStorage akan bertahan walaupun jendela browser atau tab browser ditutup, untuk menghapus datanya bisa dengan tombol “Hapus Storage” dimana tombol tersebut akan memanggil method localStorage.removeItem().

Bagaimana? Cukup mudah kan untuk menggunakan Web Storage? Selanjutnya kita akan coba terapkan localStorage pada Web Kalkulator yang kita buat untuk menampung riwayat kalkulasi yang pengguna lakukan.

### Menerapkan styling pada tabel riwayat

Sebelumnya kita sudah membuat elemen tabel yang nantinya akan menjadi tempat informasi riwayat kalkulasi, namun tampilannya masih kurang baik. Kita akan merombaknya dengan menerapkan CSS pada elemen tersebut.

Silakan buka berkas **style.css**, kemudian tambahkan rule berikut:

1. .history {
2. width: 80%;
3. margin: 30px auto 0 auto;
4. overflow: scroll;
5. }

Rule tersebut akan mengatur lebar 80% dari viewport pada elemen <div> yang menerapkan class history. Kemudian terapkan pada properti overflow dengan nilai scroll sehingga elemen <div> akan memiliki scroll bar secara horizontal dan vertikal.

Lalu tambahkan juga rule berikut pada **style.css**:

1. table {
2. border-collapse: collapse;
3. border-spacing: 0;
4. width: 100%;
5. border: 1px solid #ddd;
6. }

Rule tersebut akan membuat tabel memiliki lebar **100%** terhadap parentnya dan menetapkan border pada tepian tabelnya.

Lanjut kita tambahkan rule berikut untuk mengatur padding pada table head (<th>) dan table data (<td>) agar ukuran tampak lebih proporsional.

1. th,
2. td {
3. text-align: center;
4. padding: 16px;
5. }

Lalu kita tambahkan rule berikut untuk menerapkan backgroud-color pada element <th> dan <tr>, sementara untuk elemen <tr> kita manfaatkan **pseudo-class** dalam selectornya.

1. th {
2. background-color: orange;
3. color: white;
4. }
6. tr:nth-child(even) {
7. background-color: #d2d2d2;
8. }

Untuk mengoptimalkan tampilan pada perangkat seluler, kita atur elemen <div> untuk menerapkan width: 100% pada @media query.

1. @media screen and (max-width: 513px) {
2. .button {
3. padding: 10px;
4. }
5. .history {
6. width: 100%;
7. }
8. }

Sekarang struktur kode pada berkas **styles.css** akan tampak seperti ini:

1. \* {
2. box-sizing: border-box;
3. }
4. **body** {
5. font-family: sans-serif;
6. }
7. .flex-container-column {
8. display: flex;
9. */\* properti pendukung \*/*
10. flex-direction: column;
11. max-width: 800px;
12. margin: 0 auto;
13. text-align: right;
14. }
15. .flex-container-row {
16. display: flex;
17. }
18. .button {
19. flex-basis: 25%;
20. */\* properti pendukung \*/*
21. font-size: 1.5em;
22. text-align: center;
23. padding: 40px;
24. border: 1px solid black;
25. background: -webkit-linear-gradient(top, #d2d2d2, #ddd);
26. cursor: pointer;
27. }
28. .double {
29. flex-basis: 50%;
30. }
31. .display {
32. color: white;
33. width: 100%;
34. padding: 10px 20px;
35. background-color: #333333;
36. border: 1px solid black;
37. font-size: 2em;
38. }
39. .operator,
40. .negative,
41. .equals {
42. background: orange;
43. }
44. .card {
45. box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2);
46. border-radius: 5px;
47. padding: 30px;
48. margin-top: 20px;
49. }
50. .button:hover {
51. font-weight: bold;
52. }
53. .history {
54. width: 80%;
55. margin: 30px auto 0 auto;
56. overflow: scroll;
57. }
58. **table** {
59. border-collapse: collapse;
60. border-spacing: 0;
61. width: 100%;
62. border: 1px solid #ddd;
63. }
64. **th**,
65. **td** {
66. text-align: center;
67. padding: 16px;
68. }
69. **th** {
70. background-color: orange;
71. color: white;
72. }
73. **tr**:nth-child(even) {
74. background-color: #d2d2d2;
75. }
76. @**media** screen and (max-width: 513px) {
77. .button {
78. padding: 10px;
79. }
80. .history {
81. width: 100%;
82. }
83. }

Setelah persiapan tampilan table selesai, selanjutnya kita akan mulai menerapkan JavaScript untuk mengelola data riwayat kalkulasinya