Javascript Fundamental

Gambaran Umum

Javascript adalah suatu **bahasa pemprograman** yang sangat **umum** digunakan. Materi ini akan mencakup hal-hal dasar dalam Javascript yang nanti akan sangat berguna untuk memahami Javascript.

Tujuan Pembelajaran

1. Peserta dapat menulis *syntax Javascript* dengan benar.
2. Peserta dapat membuat sebuah program dengan menggunakan bahasa *Javascript*.
3. Peserta dapat menyimpan suatu nilai kedalam variabel.
4. Peserta dapat menentukan tipe data apa yang harus mereka gunakan untuk memecahkan suatu masalah.

Javascript?

Javascript,

Sering denger, tapi gatau itu apa?

Javascript adalah sebuah bahasa pemrograman seperti Python, C++ dan lain-lain.

Bahasa ini banyak digunakan untuk *web development.*

Temen-temen bakal banyak menemui Javascript ketika menjelajahi internet dengan menggunakan browser.

Nah, tadi **sempet disebut** tuh kalo temen-temen **bakal banyak nemuin Javascript** ketika **membuka browser**. Itu berarti bahwa **browser** mengerti bahasa **Javascript.**

Nah, tadi **sempet disebut** tuh kalo temen-temen **bakal banyak nemuin Javascript** ketika **membuka browser**. Itu berarti bahwa **browser** mengerti bahasa **Javascript** yang berarti juga, browser dapat mengeksekusi kode Javascript.

Mari kita bahas sedikit tentang bagaimana dan dimana Javascript dapat dieksekusi di slide berikutnya →

Dimanakah Javascript bisa dieksekusi?

1. **Browser**

   Setiap browser (Firefox, Chrome, Safari dan lain-lain) memiliki *engine* yang dapat mengeksekusi kode Javascript, engine ini bertugas sebagai penerjemah sekaligus penghubung antara Javascript dengan komputer kita, jadi secara garis besar, engine ini lah yang meneruskan perintah kita dan memberi tahu ke komputer apa yang harus dilakukan.

   Beberapa *Javascript engine* yang terkenal

   - **V8 Engine → Chrome**
   - **Spider Monkey → Firefox**

Dimanakah Javascript bisa dieksekusi?

2. **Node.JS**

   Nah, tadi kita bahas *engine* yang di Browser nih, sekarang kita bahas Node. Node adalah tempat lain dimana Javascript bisa dieksekusi, lebih spesifik lagi, kita langsung menjalankan Javascript di komputer kita secara langsung. Kalau di browser tadi, kita masih memiliki perantara untuk menjalankannya yaitu lewat browser. Kita akan bahas lebih mendetail tentang Node.JS ini dan bagai mana dia bekerja di kursus lain.

Kali ini kita akan bahas topik-topik ini yang mengenai Javascript

- Struktur Kode
- Variabel
- Tipe Data
- Operator
- Operator Logika (*Logical Operator*)

BINAR

Struktur Kode

Javascript

BINAR

Ehem,
di chapter kemaren kita udah bahas apa itu struktur kode lewat CSS, masih ingat?

Oke, kita ingat-ingat sedikit.

Temen-temen bisa lihat, kode disamping. Kode di samping itu merupakan kode CSS yang pernah kita bahas di chapter 1. Disitu terdapat struktur, bisa gak kira-kira temen-temen pecah satu persatu bagian yang ada di kode tersebut?

Nah, hal-hal semacam itulah yang kita sebut dengan struktur kode.

Tapi,

Struktur kode di Javascript lebih rumit dibandingkan CSS, kenapa?

Karena

Javascript adalah **sebuah bahasa pemrograman** yang berarti bisa **mengerjakan lebih banyak logika** dibandingkan **CSS** yang **hanyalah sebuah skrip**.

BINAR

Kita akan bahas beberapa hal ini yang terkait dengan struktur kode

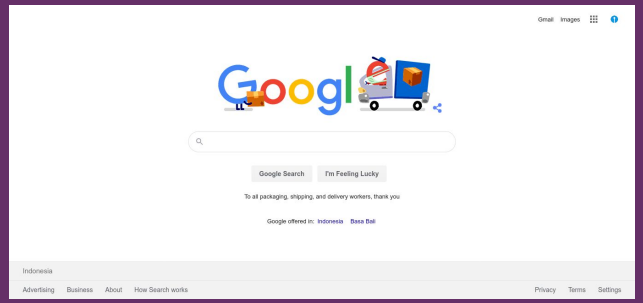
- *Statement* (Pernyataan)
- *Semicolon* (Titik Koma)
- *Comment* (Komentar/Kode yang dihiraukan) → Kita pernah bahas ini di chapter 1 HTML dan CSS

Statement (Pernyataan)

Layaknya sebuah pernyataan pada bahasa, pernyataan berfungsi untuk menyatakan suatu perintah atau sekedar ngasih info ke komputer.

Sebagai contoh, kita menyuruh komputer untuk mati, dengan cara menekan tombol shutdown di laptop kita. Nah ketika kita menekan tombol shutdown, komputer akan menjalankan suatu perintah yang sudah dibungkus dalam sebuah kode, nah untuk memberitahu perintah tersebut, maka kita harus menambahkan pernyataan di dalam kode tersebut.

Mungkin kita bisa memulai dengan *statement* “Hello World”



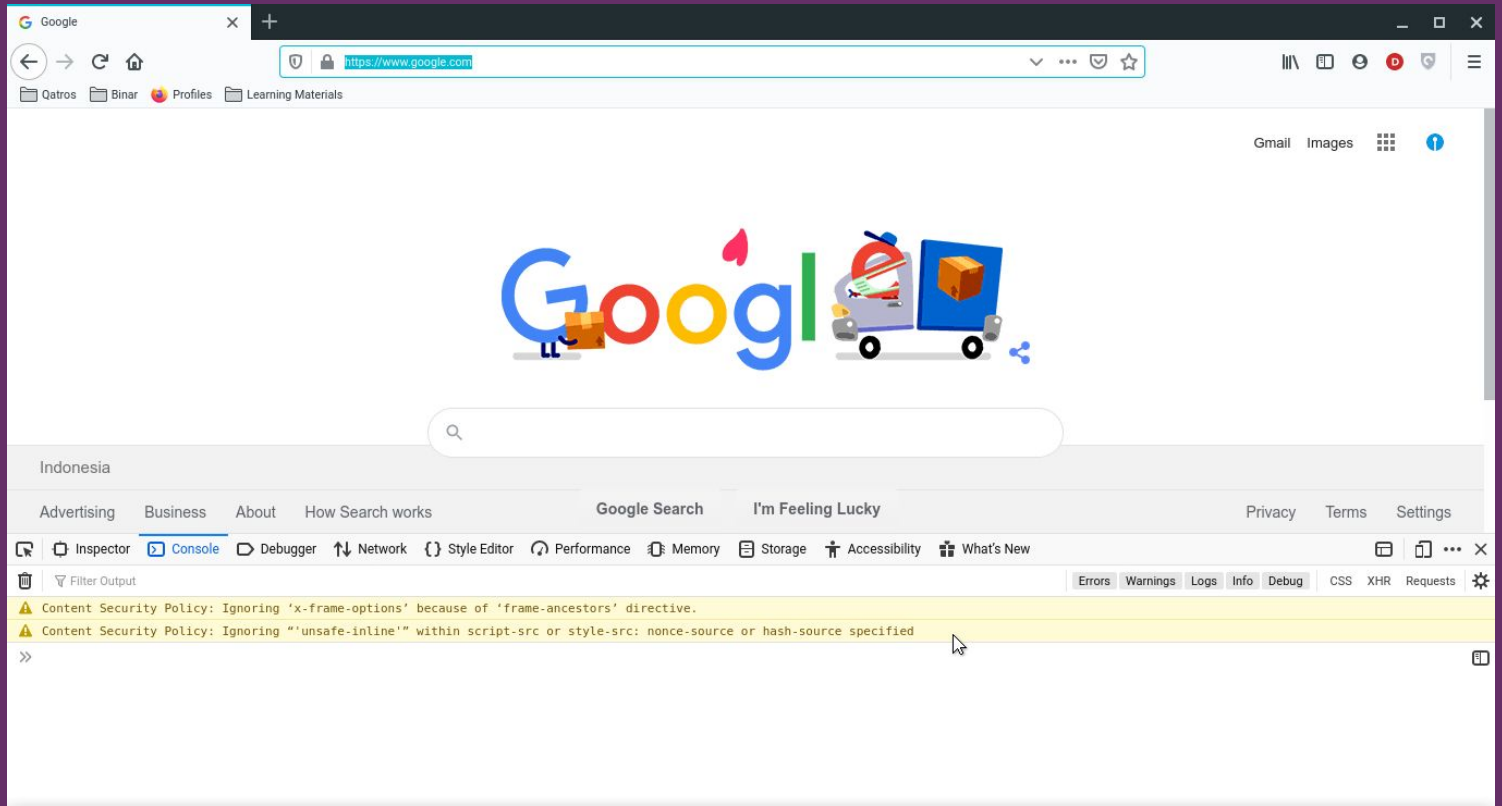
Silahkan buka browser kalian

Lalu tekan **F12**

F12

BINAR

Maka akan muncul Developer Console

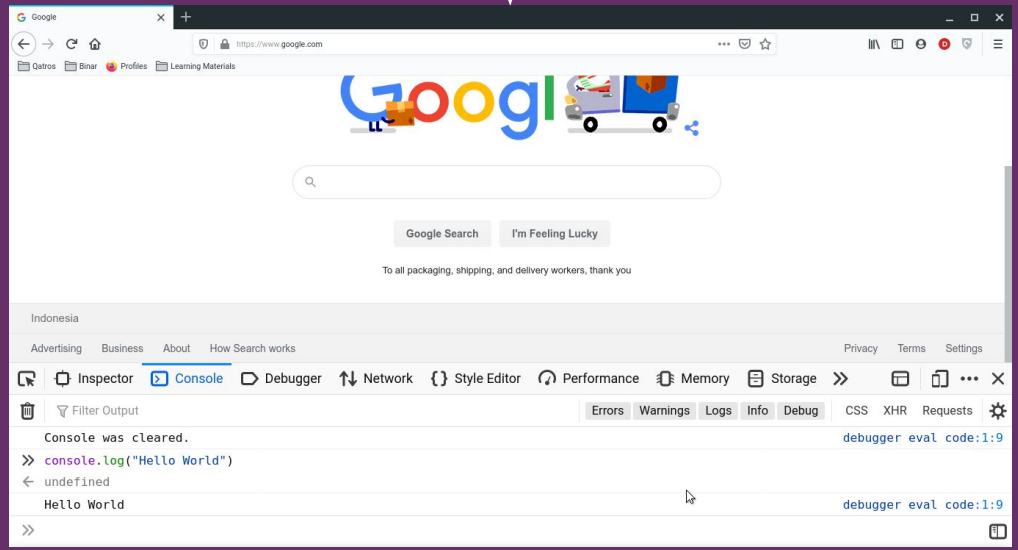


Lalu temen-temen bisa mulai ketik

```
>> console.log("Hello World")
```

di console tab yang sudah dibuka tadi terus tekan **Enter**

Maka hasilnya akan seperti ini



Hmmm,

Apa sih yang barusan kita lakuin?

Jadi yang kalian lakuin tadi adalah membuat **suatu pernyataan** kepada Browser untuk **menampilkan kata Hello World** di dalam konsol

```
>> console.log("Hello World")
```

Nah, **`console.log`** inilah yang kita sebut sebagai ***statement***. Kayak bahasa yang sering kita gunakan sehari-hari, kita bisa membuat banyak pernyataan dalam suatu program.

Untuk memisahkan pernyataan satu dengan yang lain, kita bisa menggunakan ***semicolon*** (Titik koma → **;**) atau dengan **Enter.**

```
» console.log("Pernyataan 1"); console.log("Pernyataan 2")
← undefined
  Pernyataan 1
  Pernyataan 2
```

Contoh kalo pake titik koma

```
» console.log("Pernyataan 1")
  console.log("Pernyataan 2")
← undefined
  Pernyataan 1
  Pernyataan 2
```

Contoh kalo pake **Enter**

Kalo kalian pengen ngelakuin **enter** di ***console tab,*** kalian harus pake **Shift + Enter** untuk ngelakuin itu.

Hmmm,

Apakah *statement* itu cuma `console.log` aja?

Tentu saja tidak

`console.log` hanyalah sebagai contoh saja bagaimana kita membuat suatu pernyataan di Javascript.

Semicolon (Titik Koma)

Titik koma seperti yang kita sudah bahas tadi, berfungsi untuk memisahkan *statement* satu dengan yang lainnya.

Namun, titik koma sebenarnya tidaklah wajib dituliskan di Javascript untuk pembeda antar *statement*.

Kalo kalian perhatikan disini, kita tidak perlu menambahkan titik dua kalau kita membedakan statement dengan memakai **jeda garis baru** atau biasa kita sebut dengan **implisit**. Tapi gapapa juga kok kalo kalian pengen nambahin titik dua di belakang *statement*, Javascript gaakan marah **:)**

```
>> console.log("Pernyataan 1")
   console.log("Pernyataan 2")
<- undefined
   Pernyataan 1
   Pernyataan 2
```

Contoh kalo ga pake **titik koma**

```
>> console.log("Pernyataan 1");
   console.log("Pernyataan 2");
<- undefined
   Pernyataan 1
   Pernyataan 2
```

Contoh kalo pake **titik koma**

BINAR

Tapi,

Kalo kalian pengen nulis kedua *statement* dalam satu baris, maka wajib hukumnya pake titik koma untuk memisah *statement* satu dengan yang lainnya.

```
>> console.log("Pernyataan 1"); console.log("Pernyataan 2")
<- undefined
   Pernyataan 1
   Pernyataan 2
```

Contoh kalo pake titik koma

```
>> console.log("Pernyataan 1") console.log("Pernyataan 2")
   SyntaxError: unexpected token: identifier [Learn More]
```

Contoh kalo ga pake titik koma, dan **error**

BINAR

Ada kasus lain nih,

Ga selamanya, garis baru itu dianggap *statement* baru, kalau kita membuat suatu ekspresi dan belum selesai di baris itu, maka Javascript akan terus melihat garis di bawahnya sampai ekspresi itu selesai.

```
» console.log(6
  + 10
  );
← undefined
  16
```

6 + 10

Adalah sebuah ekspresi, maka Javascript akan memindai ekspresi tersebut sampai selesai.

Berbicara soal ekspresi, kita tidak boleh meletakkan titik koma di tengah-tengah ekspresi, karena itu akan membuat Javascript bingung dalam memahami ekspresi tersebut

```
» console.log(6;
  + 10
  );
SyntaxError: missing ) after argument list [Learn More]
```

BINAR

Comment (Komentar)

Sama seperti di HTML dan CSS kemaren, komentar adalah sederet kode yang dihiraukan oleh *engine* yang menjalankannya. Nah komentar itu penting, untuk memberi tahu *developer* lain atau mungkin mengingatkan kita sendiri tentang kode yang kita tulis.

Untuk membuat komentar di Javascript, ada dua cara.

1. Komentar *One-Line* (Satu Baris)
2. Komentar *Multi-Line* (Banyak Baris)

BINAR

Komentar One-Line

Jenis komentar ini dimulai dengan dua garis miring "**/ /**".

```
// Kode ini berguna untuk nge-hack NASA
console.log("Hack NASA")
undefined
Hack NASA
```

Komentar *One-Line*

```
console.log("Bikin Google"); // Kode ini berfungsi untuk bikin startup bernama Google.
undefined
Bikin Google
```

Komentar *One-Line* mengikuti *statement*

BINAR

Komentar Multi-Line

Jenis komentar ini dimulai dengan garis miring dan bintang "**/*" dan diakhiri dengan menggunakan bintang dan garis miring "***/**".

```
/* Pernyataan 1, hanyalah pernyataan biasa, console.log */
console.log("Pernyataan 1");
undefined
Pernyataan 1
```

Komentar *Multi-Line*

```
/* Kode ini dibikin oleh saya,
   dibikin pada tanggal 2 Juli 2019.
   Fungsi kode ini buat nge-hack NASA
   */
console.log("Hack NASA");
undefined
Hack NASA
```

BINAR

Variabel

Javascript

BINAR

Temen-temen,
masih ingat pelajaran matematika?
Kita pernah diajarkan konsep variabel.
Kita akan pelajari konsep tersebut
dari sudut pandang pemrograman.

Dalam dunia pemrograman, pasti kita butuh yang namanya data. Variabel adalah salah satu tempat kita untuk menyimpan informasi data.

BINAR

Contoh

Temen-temen punya data di dalam KTP, nah di dalam KTP temen-temen pasti ada nama kalian, sebagai contoh Sabrina. Sabrina ini adalah data, dan **nama** adalah variabel. KTP menyimpan data yang nilainya **Sabrina** ke dalam sebuah tempat yaitu variabel yang bernama **nama.**

Oke,

Terus caranya bikin variabel di Javascript gimana?

BINAR

Kita buka konsol kita lagi.

```
» let pesan;
```

Kode diatas adalah sebuah *statement* yang berfungsi untuk membuat suatu variable yang bernama **pesan.**

```
» let pesan;
← undefined
» pesan = "Hello World";
← "Hello World"
```

Karena kita sudah membuat sebuah variabel, kita bisa memasukkan data ke dalamnya dengan cara seperti diatas.

```
» let pesan;
← undefined
» pesan = "Hello World";
← "Hello World"
» console.log(pesan);
← undefined
  Hello World
```

Ketika kita mencoba menjalankan console.log(pesan) maka hasil yang keluar adalah data yang kita masukkan tadi. Ini adalah bukti bahwa datanya sudah **tersimpan** di dalam variabel **pesan**

BINAR

Cara lain bikin variabel dan masukin data

```
>> let pesan = "Hello World"; // Bikin variabel sekaligus memasukkan datanya (nilai)
   console.log(pesan);
```

Kita bisa bikin beberapa variabel sekaligus, lho!

```
>> let pengguna = 'Didit', umur = 23, pesan = 'Hello';
```

```
>> // Atau biar lebih enak dibaca
   let pengguna = 'Didit'
     , umur = 23
     , pesan = 'Hello';
```

BINAR

Masih

Bingung?

BINAR

Kita dapat dengan mudah memahami konsep "variabel" jika kita membayangkannya sebagai "kotak" untuk data, dengan stiker yang diberi nama unik di atasnya.

Misalnya, pesan variabel dapat dibayangkan sebagai kotak berlabel "pesan" dengan nilai "Halo!" di dalamnya

Kita dapat memberikan nilai apapun kedalam kotak. Kita juga dapat mengubahnya sebanyak yang kita inginkan

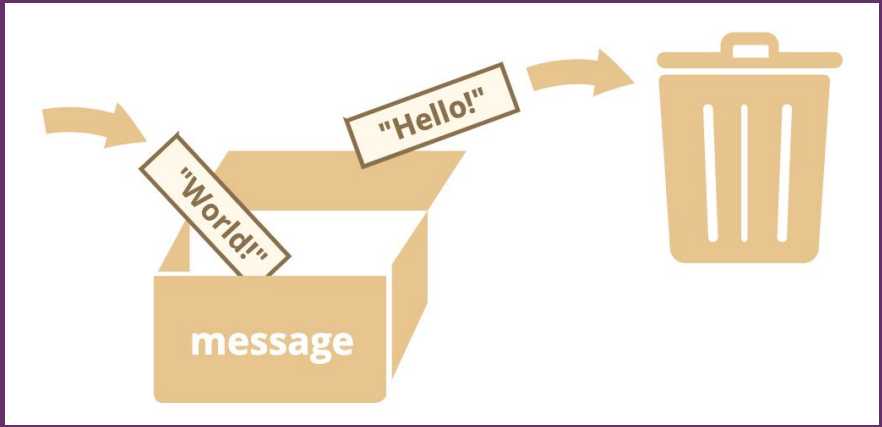
```
let pesan;

pesan = 'Helo';

pesan = 'World'; // nilai diganti

console.log(pesan);
```

Ketika nilai berganti, data sebelumnya akan dihapus dari variabel



Kita juga dapat mendeklarasikan dua variabel dan mengkopi data dari variabel yang satu ke variabel yang lainnya.

```
let halo = 'Halo dunia!';

let pesan;

// copy 'Halo dunia' dari halo ke pesan
pesan = halo;

// sekarang dua vcariabel menampung data yang sama
console.log(halo); // Halo dunia!
console.log(pesan); // Halo dunia!
```

BINAR

Penamaan Variabel

Tentu saja kita tidak bisa sembarangan dalam menamakan variabel, Di Javascript sendiri ada dua aturan dalam penamaan variabel

1. Nama harus mengandung huruf, angka atau simbol **$** dan **_**
2. Karakter pertama tidak boleh angka

BINAR

Contoh penamaan variabel yang valid

```
let namaPengguna;
let didit123;
```

Kalian bisa lihat variabel namaPengguna, disitu terdiri dari dua kata. Nah untuk Javascript, kalau kalian pengen buat variabel dengan dua kata, maka harus digabung dengan kaidah **camelCase.**

Yang menarik adalah tanda " $ " dan " _ " juga dapat digunkan dalam penamaan variabel karena kedua simbol ini merupakan simbol biasa sama seperti huruf dan tanpa makna khusus.

```
let $ = 2; // pendeklarasian variabel dengan "$"
let _ = 4; // pendeklarasian variabel dengan "_"

console.log($ + _)
```

BINAR

Contoh penamaan variabel yang gak valid

```
let 1a; // tidak bisa di awali dengan angka

let nama-saya; // tanda strip " - " tidak dapat digunakan
```

Berdasarkan aturan tadi, nama variabel tidak boleh diawali dengan angka, dan tidak boleh memakai simbol selain **$** atau **_**. Teman-teman bisa coba sendiri membuat variabel dengan nama seperti diatas.

BINAR

Variabel

Konstan

BINAR

Variabel Konstan

adalah variabel yang nilainya (atau datanya) tidak dapat dirubah-rubah seperti apapun kondisinya. Karena itulah ia dinamakan konstan

BINAR

```
» /* Kalo tadi kita bikin variabel pake kata "let"
   Nah, kalo kita pengen bikin variabel konstan,
   kita pake kata "const" */
const bumi = "bulat";
```

Dalam mendefinisikan variabel konstan, kita diwajibkan untuk langsung memasukkan nilai dari variabel tersebut

BINAR

Karena Konstanta Maka

```
>> /* Kalo tadi kita bikin variabel pake kata "let"
      Nah, kalo kita pengen bikin variabel konstan,
      kita pake kata "const" */
   const bumi = "bulat";
<- undefined
>> bumi = "datar";
(!) TypeError: invalid assignment to const `bumi' [Learn More]
```

Dilarang keras untuk merubah nilai dari variabel tersebut

Ketika seorang anggota team programer yakin bahwa suatu variabel tidak akan pernah berubah, anggota team tersebut dapat mendeklarasikannya dengan **const** untuk menjamin kemudian menginformasikan tentang hal itu dengan jelas ke semua anggota team supa tidak terjadi miss komunikasi.

BINAR

Uppercase Constants

Ada banyak kasus yang lebih baiknnya kita menggunkan konstanta sebagai variabel contohnya penamaan alias sehingga kita lebih mudah untuk mengingat alias-alias tersebut.

Variabel konstanta seperti itu biasanya diberi nama dengan menggunakan garis bawa untuk spasi.

Mari kita buat konstanta untuk warna dalam format yang disebut "web" (heksadesimal)

```
const WARNA_MERAH = "#F00";
const WARNA_HIJAU = "#0F0";
const WARNA_BIRU = "#00F";
const WARNA_ORANGE = "#FF7F00";

// ... ketika kita butuh mengambil warna maka
let warna = WARNA_BIRU;
console.log(warna); // #00F
```

Kelebihan menggunakan **variabel konstan**

- **`WARNA_ORANGE`** lebih gampang untuk diingat daripada kodenya yaitu **`#FF7F00`**
- Akan jauh lebih mudah untuk typo ketika mengetik "**`#FF7F00`**" daripada **`WARNA_ORANGE`**
- Saat membaca kode, **`WARNA_ORANGE`** jauh lebih bermakna daripda "**`#FF7F00`**"

BINAR

Tipe Data

Javascript

Mungkin temen-temen sudah mengenal sedikit tentang tipe data di Javascript, karena kalau temen-temen sadari, kalian tadi udah berjumpa sama beberapa tipe data, ketika teman-teman belajar soal *statement* sama variabel.

BINAR

Bentar-bentar,

Tipe data itu apa sih?

Tadi kalian udah tau apa itu data. Nah, data itu ada banyak jenisnya, temen-temen. Ketika temen-temen bikin *statement*

```
console.log("Hello World");
```

Disitu terdapat tipe data yang bernama **String.**

Yang mana yang string?

`"Hello World"` dengan kutip dua.

Berikut tipe data yang ada di Javascript

- String
- Number
- Boolean
- null
- undefined
- Object

BINAR

String

Tipe data yang menyimpan nilainya adalah karakter, mulai dari huruf, angka, simbol dan sebagainya, dimana nilainya akan dikelilingi oleh tanda kutip.

```
// Pake tanda kutip dua
"Ini string";
"Ini string tapi pake karakter aneh-aneh ()*@!)(^&^%^%#*";
"Ini string pake angka 123456";

// Pake tanda kutip satu
'Ini string tapi aku bisa nambahin kutip dua di string ini, liat nih ", tuh kan!';
'Gapapa kan kalo pake kutip satu?';

// Pake backtick
`Ini string tapi petiknya kebalik`;
`${10 + 11}`;
```

BINAR

Template Literal

Di dalam string, kita bisa memanggil variabel yang sudah kita deklarasikan lho, dan juga kita bisa bikin suatu ekspresi didalamnya.

String yang semacam ini lah yang kita sebut dengan ***template literal***.

Template Literal wajib memakai ***backtick*** (` ). Untuk menambahkan suatu kode yang akan dieksekusi di dalam string, kita menambahkan karakter seperti ini `${...}`. Nah kode yang ada di dalam karakter itu akan dievaluasi oleh Javascript (Dijalankan).

```
>> let nama = "Sabrina";
   let pekerjaan = "Fasilitator Binar";

   console.log(`Hi, nama aku ${nama} dan pekerjaanku sekarang adalah ${pekerjaan}`);
<- undefined
   Hi, nama aku Sabrina dan pekerjaanku sekarang adalah Fasilitator Binar
```

BINAR

Boolean

Tipe data yang hanya memiliki dua nilai saja, ***true*** atau ***false***. Tipe data boolean ini akan sangat sering dipakai pada pengecekan suatu data.

```
let apakahSedangTerjadiPandemi = true;
let apakahAkuHarusKeluarRumah = false;
```

BINAR

Nilai Boolean bisa saja datang dari suatu perbandingan

Perbandingan disini yang dimaksud adalah ***Logical Operator***, yang nanti akan kita bahas di dalam kursus ini.

```
>> let apakah1LebihDari2 = 1 > 2;
   console.log(apakah1LebihDari2);
<- undefined
   false
```

```
>> let a = 1;
   let b = 2;
   let apakahASamaDenganB = a == b;
   console.log(apakahASamaDenganB);
<- undefined
   false
```

BINAR

Null

Tipe data yang nilainya kosong. Dia hanya mewakili kekosongan atau tidak ada.

Pada pemrograman javascript null bukanlah sebuah referensi ke objek yang tidak ada atau "null pointer" seperti dalam beberapa bahasa pemrograman lain. Dalam *conditional* data ini akan dianggap ***false***.

```
» let kosong = null;
  console.log(kosong);
```

BINAR

Undefined

Tipe data tidak terdefinisi, artinya tidak ada sumber nilai pada data tersebut. Nilai ini biasanya muncul ketika kita mendefinisikan variabel tetapi kita tidak memasukkan nilai ke dalamnya sama sekali. Dalam *conditional* data ini akan dianggap ***false***.

```
>> let tidakTerdefinisi;
<- undefined
>> console.log(tidakTerdefinisi);
<- undefined
   undefined
```

BINAR

Object

Tipe data sebelumnya, kita sebut dengan tipe data “Primitif”, karena mereka secara langsung merepresentasikan suatu nilai. Namun berbeda dengan *Object*, *object* adalah suatu tipe data yang menyimpan koleksi tipe data yang lain. Contoh, suatu object bisa memiliki string, number, boolean sekaligus. Karena object adalah koleksi dari data, maka harus ada kunci untuk memanggil salah satu koleksi data di dalam object, nah kunci tersebut biasa kita sebut dengan ***key,*** dan nilainya biasa kita sebut dengan ***value***, dan sepasang ***key*** dan ***value*** kita sebut dengan ***property***.

Kalian bisa lihat,

Object dapat menyimpan berbagai jenis tipe data yang ada, dan memetakannya dengan menggunakan key.

```
>> let contohObjek = {
     nama: "Sabrina",
     umur: 25,
     jenisKelamin: "Wanita",
     sudahMenikah: false
   }
```

Yang kita sebut dengan key adalah,

nama, umur, jenisKelamin, sudahMenikah

Teman-teman bisa cek tipe data yang ada dengan menggunakan operator **`typeof.`**

Typeof ini ada dua cara dalam menulis syntaxnya.

1. Sebagai operator → `typeof x`
2. Sebagai *function* → `typeof(x)`

```
>> typeof("Hello World");
<- "string"
```

```
>> let x = "Hello World";
<- undefined
>> typeof x;
<- "string"
>> typeof 123
<- "number"
>> typeof true;
<- "boolean"
>> typeof {
     name: "Sabrina"
   };
<- "object"
>> typeof null;
<- "object"
>> typeof undefined;
<- "undefined"
```

BINAR

Operator

Javascript

Kita tahu banyak operator dari sekolah, seperti penjumlahan (+), perkalian (*), pengurangan (-), dan operator-operator lainnya.

Kali ini kita akan fokus untuk membahas aspek operator yang tidak kita dapatkan dibangku sekolah.

Unary, Binary dan Operan

Sebelum kita bahas lebih detail, ada baiknya kita pahami dulu istilahnya.

- **Operan**

  Operan adalah pelaku dari suatu operasi. Maksudnya apa, coba perhatikan contoh berikut.

  1 + 4

  Nah di dalam operasi di atas, terdapat dua operan. Yaitu operan kanan, 1, dan operan kiri, 4.

- **Unary**

  Unary adalah suatu operasi yang hanya memiliki 1 operan.

```
>> let x = 1;
   x = -x;
   console.log(x) // Menggunakan unary negation
```

-x adalah unary

- **Binary**

  Binary adalah suatu operasi yang memiliki lebih dari 1 operan..

```
>> let x = 10;
   let y = 5;
   console.log(x - y); /*
     Terjadi operasi di dalam console.log()
     Dimana x - y, x adalah operan dan y adalah operan
     Operasi ini kita sebut dengan binary
   */
```

Binary + pada rangkaian string

Ternyata, operator plus itu ga cuma bisa dipake dalam operasi aritmatika aja. Melainkan bisa digunakan untuk melakukan *concatenation* (Perangkaian) sebuah string.

```
» let a = "Hello";
  let b = "World";
  console.log(a + b);
← undefined
  HelloWorld
```

Biasanya operator plus (+) digunakan untuk menjumlahkan angka.Tetapi pada biner plus (+) diterapkan pada String, + bukan untuk menjumlahkan tetapi digunakan untuk menyatukan

```
» /* Jika salah satu
     operan adalah string,
     maka operan yang
     lain akan dianggap string juga */

  console.log("1" + 2);
← undefined
  12
```

Perhatikan jika salah satu operan adalah string, maka operan yang lain akan ikut dikonversi menjadi string juga

```
» /* Namun berbeda juga
     kalau di dalam operasi itu
     terdapat operan yang bisa melakukan
     operasi aritmatika.
     Javascript akan menjalankan
     operasi tersebut terlebih dahulu baru
     menambahkannya ke string */
  console.log(4 + 4 + "2"); // Hasilnya bukan "442"
← undefined
  82
```

Namun, perhatikan bahwa operasi berjalan dari kiri ke kanan. Jika ada dua angka diikuti oleh string, angka-angka itu akan ditambahkan sebelum dikonversi ke string

Numeric conversion, unary +

Plus (+) ada dalam dua bentuk: bentuk binary yang kita gunakan di slide sebelumnya dan bentuk unary. Unary plus atau dengan kata lain operator plus (+) diterapkan pada nilai tunggal, tidak melakukan apapun pada angka. Tetapi jika operan bukan angka, plus unary mengubahnya menjadi angka.

```
// Tidak berpengaruh pada angka
let x = 1;
console.log( +x ); // 1

let y = -2;
console.log( +y ); // -2

// Mengkonversi yang bukan angka
console.log( +true ); // 1
console.log( +"" );   // 0
```

Mengonversi string menjadi number, sebenarnya bisa kita lakukan dengan menggunakan function **`Number("12345")`**, tapi cara diatas lebih singkat untuk melakukannya.

Javascript - Operators : Number Conversion

Konversi dari string menjadi number ini akan sangat sering kalian jumpai ketika kalian membuat form HTML.
Karena akan sangat memungkin bahwa hasil input dari form HTML itu datanya akan berupa string.
Berikut contoh bagaimana cara melakukan operasi aritmatika dengan data yang berupa string

```
let apel = "2";
let mangga = "3";

console.log( apel + mangga ); // "23", string binar plus digabungkan
```

Contoh yang salah

Jika kita mau menggunakannya sebagai angka, kita perlu mengonversi dan menjumlahkannya

```
let apel = "2";
let mangga = "3";

// kedua nilai dikonversi menjadi angka sebelum buler plus
console.log( +apel + +mangga ); // 5

// variasi yang panjang
// console.log( Number(apel) + Number(mangga) ); // 5
```

Contoh yang benar

BINAR

Tingkatan Prioritas Operasi

Sama seperti matematika yang diajarkan di sekolah, Setiap operasi memiliki tingkat prioritas mana yang harus dikerjakan terlebih dahulu.

```
2 + 2 * 10
```

Dari operasi diatas, dapat kita pastikan bahwa operasi yang akan dijalankan terlebih dahulu adalah `2 * 10`

Tapi,

Kalo dalam hal
Programming gimana?

Tabel dari tingkatan prioritas dari tiap-tiap operasi

Ada banyak operator di bahasa pemrograman javascript, setiap operator memiliki urutan prioritas yang sesuai. Angka yang tertinggi akan dieksekusi terlebih dahulu dan jika prioritas pengurutannya adalah sama maka urutan eksekusi dilakukan dari kiri ke kanan.

Prioritas	Nama	Tanda
17	unary plus	+
17	unary negation	-
15	perkalian	*
15	pembagian	/
13	penambahan	+
13	pengurangan	-
3	sama dengan	=

Seperti yang kita lihat "unary plus" memiliki prioritas 17 yang lebih tinggi dari 13 "penambahan" (binary plus), itulah sebabnya dalam ekspresi seperti "`+apel + +mangga`", plus unary berfungsi sebelum penambahan.

BINAR

Assignment (Sama Dengan)

Penentuan nilai dari sebuah data.

Ketika kita bilang `let a = 1`, maka kita sudah melakukan sebuah *assignment* terhadap variabel a.

Hal ini berarti, kita memberi tahu bahwa variabel a bernilai 1.

```
let x = 2 * 2 + 1;

console.log( x ); // 5
```

Memungkinkan juga kalau kita ingin melakukan *assignment* berantai.

```
let a, b, c;

a = b = c = 2 + 2;

console.log( a ); // 4
console.log( b ); // 4
console.log( c ); // 4
```

Assignment berantai mengevaluasi dari kanan ke kiri. Pertama, ekspresi paling kanan 2 + 2 dievaluasi kemudian ditetapkan ke variabel di sebelah kiri: c, b dan a. Pada akhirnya, semua variabel berbagi nilai tunggal.

BINAR

Eksponensial (Perpangkatan)

Operator eksponensial ** adalah tambahan terbaru untuk bahasa pemrograman.

Untuk bilangan asli b, hasil dari ** adalah dikalikan dengan sendirinya b kali.

```
console.log( 2 ** 2 ); // 4  (2 * 2)
console.log( 2 ** 3 ); // 8  (2 * 2 * 2)
console.log( 2 ** 4 ); // 16 (2 * 2 * 2 * 2)
```

Increment dan *Decrement* (Peningkatan atau Penurunan)

Menaikkan satu angka atau menurunkan satu angka dari suatu variabel yang nilainya adalah angka

Increment

```
let hitung = 2;
hitung++;        // bekerja sama dengan penghitung = penghitung + 1, tetapi lebih pendek
console.log( hitung ); // 3
```

Decrement

```
let hitung = 2;
hitung--;        // bekerja sama dengan penghitung = penghitung - 1, tetapi lebih pendek
console.log( hitung ); // 1
```

PENTING: Operator ini hanya berlaku untuk variabel saja, kalo kalian menambahkan operator ini pada sebuah angka secara langsung, maka akan menyebabkan error

Javascript - Operators : Increment and Decrement

BINAR

Operator ++ dan -- dapat ditempatkan sebelum atau sesudah variabel.

Ketika operator berada setelah variabel maka operator akan berbentuk "posifix form": hitung++ dan sebaliknya jika operator berada di awal variabel makan akan berbentuk "prefix form": ++hitung.

Kedua pernyataan ini melakukan hal yang sama yaitu meningkatkan (increase) hitung dengan 1.

Apakah ada perbedaan? ya, tetapi kita hanya akan melihat perbedaan tersebut jika kita menggunakan nilai yang dikembalikan dari ++ / -.

Mari kita perjelas. Seperti yang kira ketahui, semua operator mengembalikan nilai. Prefix form mengembalikan nilai baru sementara bentuk posfix form akan mengembalikan nilai lama sebelum dilakuakn incresing / decreasing.

Untuk melihat perbedaannya, mari kita lihat contoh dibawah

```
let hitung = 1;
let a = ++hitung; // (*)

console.log(a); // 2
```

```
let hitung = 1;
let a = hitung++; // (*) mengubah ++hitung ke hitung++

console.log(a); // 1
```

Bitwise

Operator Bitwise memperlakukan argumen sebagai angka integer 32-bit dan bekerja pada level reprentasi binernya.

Operator ini tidak spesifik hanya untuk javascript, tetapi juga mendukung sebagian besar bahasa pemrograman.

Berikut beberapa contoh operator yang tersedia

- `AND ( & )`
- `OR ( | )`
- `XOR ( ^ )`
- `NOT ( ~ )`
- `LEFT SHIFT ( << )`
- `RIGHT SHIFT ( >> )`
- `ZERO-FILL RIGHT SHIFT ( >>> )`

Operator ini sangat jarang digunakan. Untuk memahaminya, kita perlu mempelajari representasi *low-level number* dan tidak akan optimal jika kita melakukannya sekarang karena kita tidak akan membutuhkannya dalam waktu dekat ini. Jika anda penasaran, anda dapat membaca artikel Operaror Bitwise MDN. Akan lebih praktis untuk melakukannya ketika ada kebutuhan nyata.

BINAR

Modify-in-place

Kita sering membutuhkan penerapan operator ke variabel dan menyimpan hasil baru dalam variabel yang sama juga.

```
let n = 2;
n = n + 5;
n = n * 2;
```

Kita mempunyai variabel n yang ingin kita tambahkan dan kalikan dengan beberapa angka, di baris kedua berarti 2 + 5, dimana hasil dari baris kedua adalah 7, dan disimpan kembali ke variabel n

Notasi diatas bisa disingkat juga menjadi seperti ini

```
let n = 2;
n += 5; // sekarang n = 7 (sama seperti n = n + 5)
n *= 2; // sekarang n = 14 (sama seperti n = n * 2)

console.log( n ); // 14
```

"modify-and-assign" operator berlaku juga untuk semua operator aritmatika dan bitwise: /=, -= dll.

Operator tersebut memiliki prioritas yang sama dengan prioritas normal, sehingga akan dijalankan sebagian perhitungan lainnya

```
let n = 2;

n *= 3 + 5;

console.log( n ); // 16  (rbagian kanan akan dievaluasi terlebih dahulu sama dengan n *= 8)
```

BINAR

Comma

Operator koma adalah salah satu operator yang paling langka dan paling tidak biasa. Terkadang digunakan untuk menulis kode yang lebih pendek, jadi kita perlu mengetahuinya untuk memahami apa yang akan terjadi.

Operator koma memungkinkan kita untuk mengevaluasi beberapa ekspresi, membaginya dengan koma. Masing-masing dievaluasi tetapi hanya hasil yang terakhir yang akan dikembalikan

```
let a = (1 + 2, 3 + 4);

console.log( a ); // 7 (Hasil dari 3 + 4)
```

Logical Operator

Javascript

Ada tiga logical operator dalam javascript: || (OR), && (AND), ! (NOT).

Meskipun disebut "logic" operator-operator inipun dapat diterapkan pada nilai-nilai jenis apapun tidak hanya boolean. Hasilnya juga bisa dari jenis apapun.

BINAR

|| (OR)

Operator "OR" diwakili dengan dua simbol garis vertikal

```
hasil = a || b;
```

Dalam pemrograman klasik, logika OR dimaksudkan hanya untuk memanipulasi nilai boolean. Jika salah satu argumennya benar maka akan mengembalikan nilai benar dan jika salah maka akan mengembalikan nilai salah.

```
alert( true || true );   // true
alert( false || true );  // true
alert( true || false );  // true
alert( false || false ); // false
```

&& (AND)

Operator AND ini dievaluasi dari kiri ke kanan, jika salah satu operan bernilai ***false*** maka semuanya akan bernilai dianggap **false**

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

Mari kita lihat contoh penggunaannya

```
let jam = 12;
let menit = 30;

if (jam == 12 && menit == 30) {
  alert( 'Sekarang jam 12:30' );
}
```

BINAR

! (NOT)

Operator ini berfungsi untuk membalikkan nilai boolean.

```
>> let hujan = false;
   if (!hujan) console.log("Gausah pakai payung");
   // Kalau tidak hujan gausah pakai payung
<- undefined

   Gausah pakai payung
```

BINAR

String

Tipe data yang menyimpan karakter, dan dibungkus oleh tanda kutip.

Number

Tipe data yang berupa angka dan dapat melakukan operasi aritmatik.

Boolean

Tipe data yang melakukan penegasan ya atau tidak.

Statement

Sebuah kode yang berupa pernyataan, yang biasanya berbentuk suatu aksi.

Yang akan dijalankan komputer

Semicolon

Pemisah antar statement

BINAR

Variabel

Tempat menyimpan data.

Binary +

Digunakan untuk menjumlahkan angka atau merangkai string

&&

Logical Operator dimana operasi ini dieksekusi dari kiri ke kanan, jika dia menemui operan yang bernilai ***false*** maka operan tersebut akan menjadi hasil dari operasi tersebut, dan apabila tidak ada operan yang bernilai ***false***, maka hasil dari operasi tersebut adalah operan terakhir

||

Logical Operator dimana operasi ini dieksekusi dari kiri ke kanan, jika dia menemui operan yang bernilai ***true*** maka operan tersebut akan menjadi hasil dari operasi tersebut, dan apabila tidak ada operan yang bernilai ***true***, maka hasil dari operasi tersebut adalah operan terakhir

BINAR

0, “”, null, undefined

Dalam *conditional* data ini akan dianggap ***false***.

BINAR

Referensi

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Expressions_and_Operators
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Logical_Operators

BINAR

Terima kasih