Chrome DevTools

Obyektif:

- Menggunakan DevTools sebagai tempat belajar HTML, CSS dan JavaScript
- Mengenal Panel DevTools
- Mempelajari teknik Debugging

Daftar Isi

Chrome DevTools

Chrome DevTools adalah tool (alat bantu) untuk web developer secara langsung untuk melakukan diagnosa, debugging, performance, dan lainnya. Ditujukan untuk mencari problem yang terjadi pada aplikasi, juga dengan tujuan mengoptimalkan performance web.

Langkah awal untuk mempelajari Devtools:

DevTools mempercepat pengembangan aplikasi berbasis web.
Buka DevTools dan tampilkan panel berikut:

- ☐ Device Mode
- ☐ Elements Panel
- ☐ Console Panel
- ☐ Sources Panel
- ☐ Network Panel
- ☐ Performance Panel
- ☐ Memory Panel
- ☐ Security Panel
- ☐ Application Panel

Belajar JavaScript dengan Console Panel

Aktifkan Chrome, menampilkan halaman kosong. Buka DevTools dari menu *"Hamburger"* disebelah kanan atas Chrome. Pilih **More Tools** -> **Developer Tools** . Kemudian pilih Panel **Console.**

Lakukan clear() untuk mendaptkan menu prompt seperti di gambar tersebut.

Contoh 1: Block Scope

Selanjutnya ikut input berikut ini, masukkan satu persatu, kemudian tekan Enter.

```
var a= 100;
if (a>0) {
   var a= 900;
   console.log(a);
}
console.log(a);
```

Jawab: Nilai a yang terakhir adalah ❍ 100 ❍ 900

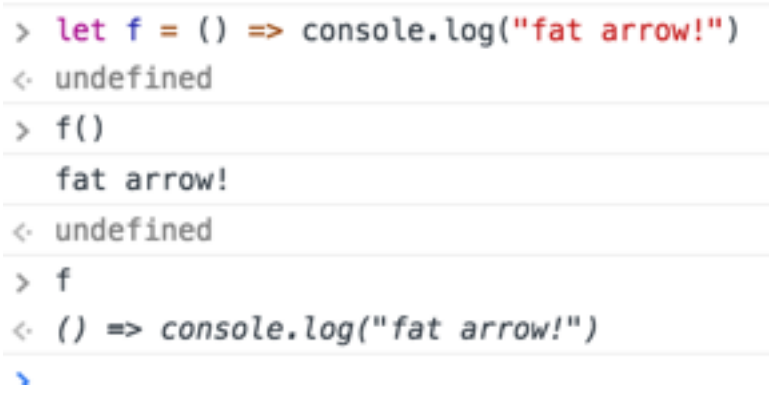
```
let b= 100;
if (b>0) {
   let b= 900;
   console.log(b);
}
console.log(b);
```

Jawab: Nilai b yang terakhir adalah ❍ 100 ❍ 900

Dari kedua program tersebut dapat dilihat bahwa deklarasi dengan mempunyai *"**Block scope***", artinya nilai variable hanya berlaku di dalam block, dimana variable tersebut didefinisikan.

Contoh 2a: Fat Arrow

Buat fungsi berikut:



f adalah sebuah fungsi yang didefinisikan melalui fat-arrow. Sebelum ada => ini, maka fungsi tersebut dideklarasikan dengan menggunakan function anonymous, artinya fungsi yang dideklarasikan tanpa nama.

```
let g = function () {
   console.log ("Fungsi Anonymous");
}
```

Fungsi ini diberikan tanpa parameter, sehingga memanggilnya cukup dengan g() .

Contoh 2b: Fat-Arrow

Buat fat-arrow dengan nama ***salam*** dari fungsi anonymous berikut:

```
let salam = function(nama) {
   console.log ( ` Hallo ${nama}, selamat jumpa dengan kami ` );
}
```

Perhatikan bahwa tanda awal yang digunakan adalah ` atau disebut sebagai "back-quote" . Kemudian kalimat diakhiri dengan tanda yang sama .
(Bahasa Indonesia : tanda kutip terbalik?)

Test fungsi tersebut:

```
> salam("Dastan")
Hallo Dastan, selamat jumpa dengan kami
```

Solusi dengan fat-arrow:

```
let salam_juga = _____________________________________________
```

Apa yang dipelajari disini adalah:

1. Bila parameter terdiri atas satu parameter saja, maka tanda kurung dapat diabaikan.
2. Bila instruksi didalam fat-arrow hanya terdiri dari 1 baris, maka kurung kurawal dapat dibaikan.

```
let myf = (a) => { return a * a }
```

menjadi:

```
let myf = a => a* a
```

Via DevTools Chrome Panel console:

```
> myf
a => a* a
> myf(12)
144
```

Contoh 3: IIFE (Immediately Invoke Function Expression)

IIFE sering digunakan sebagai inisialisasi Obyek, karena IIFE langsung meneksekusi program dan menyiapkan Obyek lengekap dengan properti dan metoda.

Sebuah fungsi di JS dapat memberikan nilai balik (return) berupa Obyek.

```
var obj = function() {
   return {
       nama: 'Badu',
       alamat: 'JL. Biawak 200',
       kota: 'Manado'
   }
}
```

Pada console dilihat hasilnya:

```
> obj
ƒ () {
   return {
       nama: 'Badu',
       alamat: 'JL. Biawak 200',
       kota: 'Manado'
   }
}
```

obj adalah fungsi yang masih harus dieksekusi dengan menggunakan tanda ().

```
> var peg= obj()
undefined
> peg
{nama: "Badu", alamat: "JL. Biawak 200", kota: "Manado"}
```

peg merupakan variable hasil eksekusi fungsi obj. Hasil tersebut adalah sebuah Obyek yang bisa dinyatakan dengan typeof.

```
> typeof peg
"object"
> peg.nama
"Badu"
> peg.kota
"Manado"
```

Dengan demikian fungsi obj tersebut harus lebih dahulu dieksekusi untuk mendapatkan obyek peg.

Cara yang lebih singkat adalah dengan langsung mengeksekusi fungsi tersebut pada saat deklarasi. Bentuk ini sama dengan ***pegawai*** = (**function()** { }) () sama dengan ***pegawai()***

```
var pegawai = (
   function() {
   return {
       nama: 'Badu',
       alamat: 'JL. Biawak 200',
       kota: 'Manado'
   }
 }
) ();
```

```
> pegawai
{nama: "Badu", alamat: "JL. Biawak 200", kota: "Manado"}
> pegawai.kota
"Manado"
```

Obyek pegawai adalah hasil eksekusi langsung fungsi anonymous, yang mempunyai properti nama, alamat, dan kota.

Untuk melengkapi obyek pegawai, dimasukkan elemen baru yaitu honor. Honor ini tidak dapat diubah dari luar, kecual menggunakan fungsi **modifyHonor()** dan dapat dibaca dengan **getHonor()**.

```
var pegawai = (
   function() {
     let honor = 5000;
     return {
       nama: 'Badu',
       alamat: 'JL. Biawak 200',
       kota: 'Manado',
       getHonor: function() { return honor },
       setHonor: function(new_honor) { honor = new_honor }
     }
   }
) ();
```

```
> pegawai.getHonor()
5000
> pegawai.setHonor(10000)
undefined
> pegawai.getHonor()
10000
```

Kedua metode tersebut merupakan fungsi anonymous dengan hanya satu baris code saja. Tapi bila membuat fungsi dengan code yang panjang, maka secara kosmetik nama properti akan menjadi sulit dideteksi.

Untuk itu sebaiknya program dipindahkan keatas, sedangkan dibawa retun { } hanya merupakan kumpulan nama properti dan metode saja.

```
var pegawai = (
   function() {
     let honor = 5000;
     function getHonor() {
         return honor;
     };
     function setHonor(new_honor) {
         honor = new_honor;
     };
     return {
       nama: 'Badu',
       alamat: 'JL. Biawak 200',
       kota: 'Manado',
       getHonor: getHonor,
       setHonor: setHonor
     }
   }
) ();
```

```
> pegawai.getHonor
ƒ getHonor() {
         return honor;
     }
> pegawai.getHonor()
5000
> pegawai.setHonor(20000)
undefined
> pegawai.getHonor()
20000
```

Debug JavaScript

Untuk melakukan Debug, dibutuhkan **program html dan javascript**, sebagai berikut : (Lihat juga [1]).

File: **add2numbers.html**

```
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>add2numbers</title>
<style>
   input {
      font-size: large
   }
</style>
</head>
<body>
<h1>Tambahkan Dua angka</h1>

<div class="calc">
  <input type="text" size="5" placeholder="0" />
   +
  <input type="text" size="5" placeholder="0" />

  <button> = </button>
  <input type="text" size="5" readonly />
</div>

<script src="add2numbers.js"></script>
</body>
</html>
```

JavaScript:

```
// add2numbers.js
function kalkulator() {
   let angka = document.querySelectorAll('input');
   let i1= angka[0].value ; //angka pertama
   let i2= angka[1].value ; //angka kedua
   angka[3].value= i1 + i2 ;
}
let tombol = document.querySelector('button');
tombol.addEventListener('click', kalkulator);
```

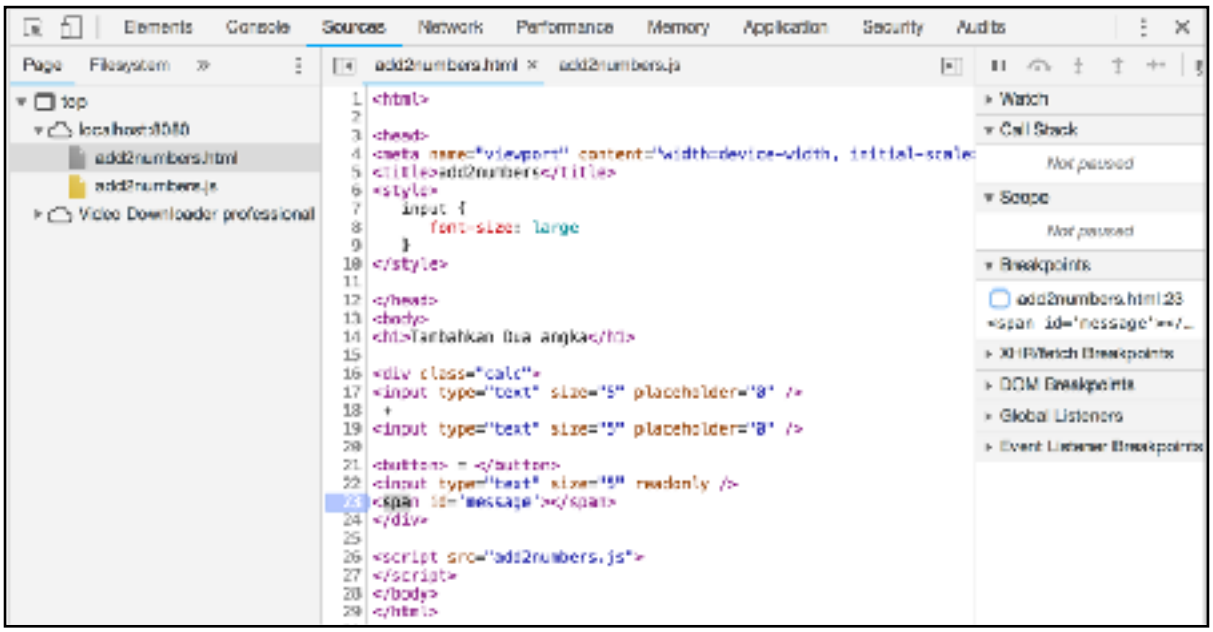
Jalankan HTTP Server untuk menampilkan file tersebut:

```
$ http-server -a localhost
```

[1] https://developers.google.com/web/tools/chrome-devtools/

Buka Chrome dengan: http://localhost:8080/add2numbers.html .

Buka DevTools dan pilih "**Sources**".



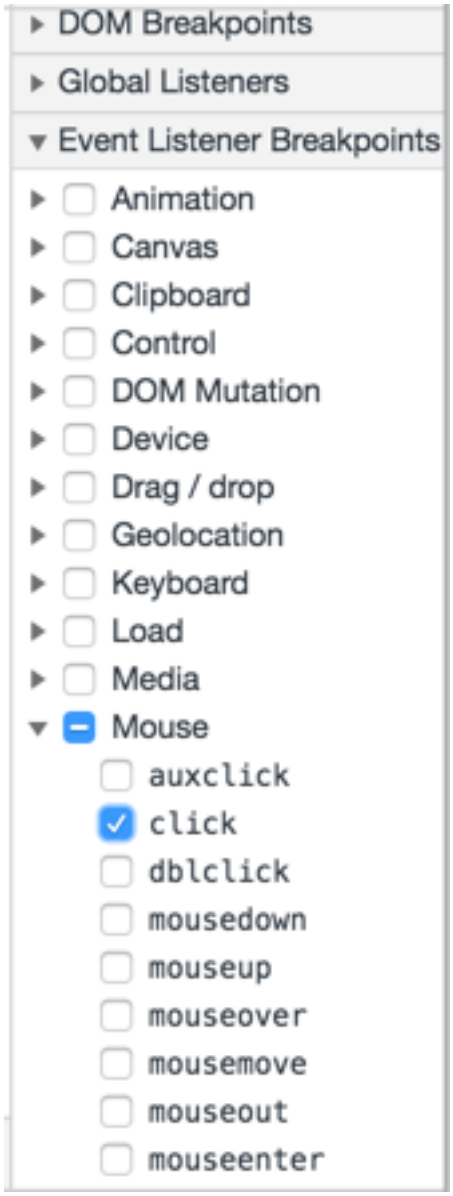
Kedua file source *.html dan *.js berada di Panel kiri. Panel tengah untuk menampilkan source-code, sedangkan Panel kanan digunakan untuk menentukan breakpoint (proses debugging).

Pada bagian kanan, pilih **Event Listener Breakpoints** .

Selanjutnya Panel akan menampilkan submenu berupa Animasi, Canvas, Clipboard, Control, dan lainnya.

Pada bagian tengah, pilih **Mouse**. Submenu akan menampilkan banyak event, diantaranya click, dblclick, mouseup, mousedown, dan seterusnya.

Untuk Debugger program ini, pilih '**click**' .



Aktifkan checkbox pada click.

Seterunya eksekusiprogram akan berhenti pada saat terjadi click, dan program ditersukan pada line berikutnya.

```
tombol.addEventListener('click', kalkulator);
```

Setelah terjadi klik, eksekusi dilanjutkan padq fungsi kalkulator( ).

```
function kalkulator() {
   let angka = document.querySelectorAll
('input');
   let i1= angka[0].value ; //angka pertama
   let i2= angka[1].value ; //angka kedua
   angka[3].value= i1 + i2 ;
}
```

Masukkan angka pada input field seperti berikut:

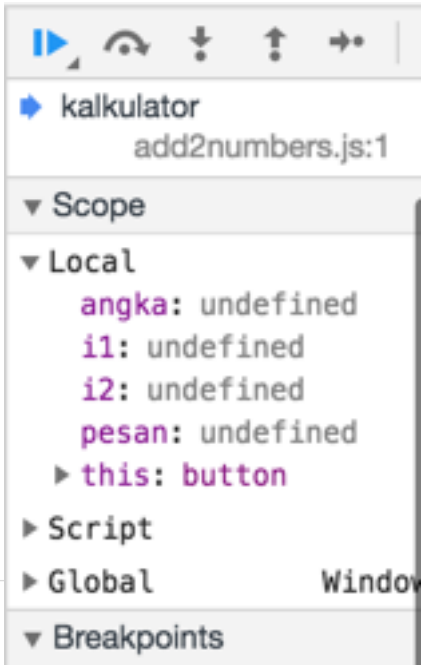
Setelah tombol '=' dipilih, maka debugger mendapatkan event 'click' dan layar berubah menjadi seperti diatas.

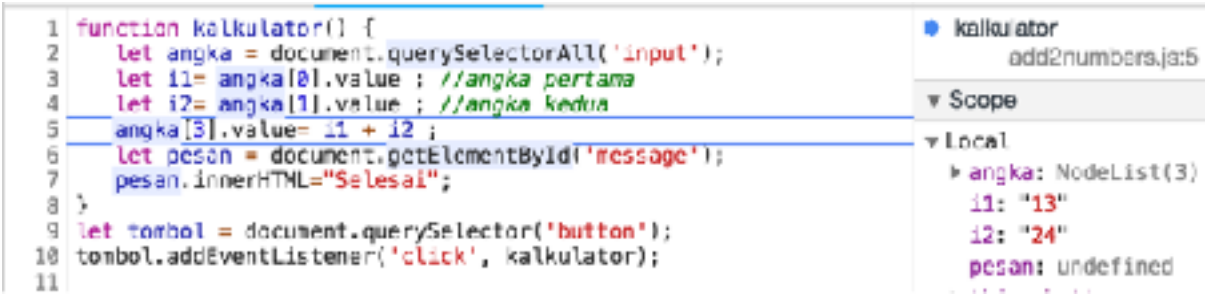
Paused in debugger menyatakan, bahwa program sedang berhenti dari eksekusi, karena breakpoint pada debugger.

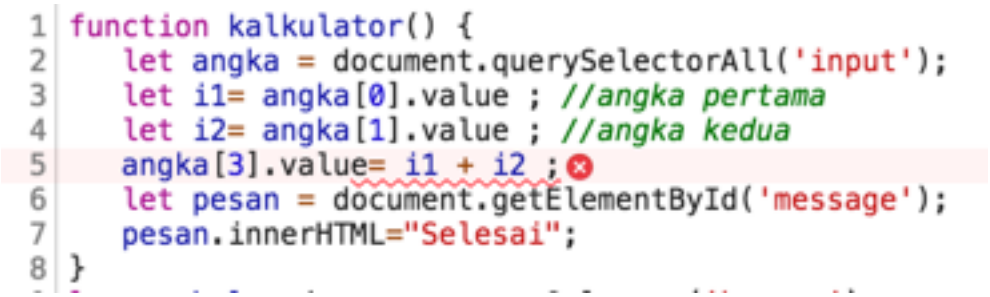
Tanda pertama adalah meneruskan program, sedang tanda kedua seperti lengkungan panah, adalah stepping ke baris berikutnya.

Sementara windows debugger memberikan informasi tentang kondisi variable angka, i1, i2, dan lainnya.

Tekan tanda panah melengkung tersebut untuk masuk

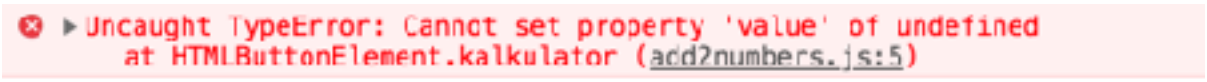






ke baris berikutnya.

Setelah langkah ke 4, maka gambar menampilkan hasil debugger, kali ini variable i1 dan i2 sudah terisi dengan string "13" dan "24" .



Step berikutnya terjadi error. Pada console terlihat pesan error:

Ternyata bahwa array **angka[3].value** adalah undefined, alias tidak ada, sehingga tidak bisa direferensikan. Perhatikan bahwa *pesan.innerHTML* adalah bersifat kosmetik semata.

```
let angka = document.querySelectorAll('input');
```

Code ini seharusnya menghasilkan array, sejumlah input yang tersedia. Input pertama dan kedua digunakan untuk memasukkan angka, sedangkan input ketiga digunakan untuk menampilkan angka (hasil penambahan).

Array yang ada adalah angka[0], angka[1] dan angka[2].

Sedangkan angka[3] tidak eksis, sehingga muncul error.

Perbaiki source-code, dan ganti *angka[3].value* dengan ***angka[2].value***.

Setelah memperbaiki source-code, ulangi proses yang sama.

Hasilnya adalah penyambungan (konkatenasi) dua buah string, bukan penambahan angka.

Pindah ke console, dan lakukan pengecekan berikut:

```
> i2
< "24"
> i1
< "13"
> i1 + i2
< "1324"
```

Seharusnya penambahan ini dilakukan bukan antar string, tapi antar integer (bilangan bulat). JS mempunya fungsi **parseInt( )** untuk mengkonversi string menjadi integer.

Dicoba pada Console:

```
> parseInt(i1) + parseInt(i2)
< 37
```

Dengan demikian program source diperbaiki sebagai berikut:

```
let i1= angka[0].value ; //angka pertama
let i2= angka[1].value ; //angka kedua
angka[2].value= parseInt(i1) + parseInt(i2) ;
```

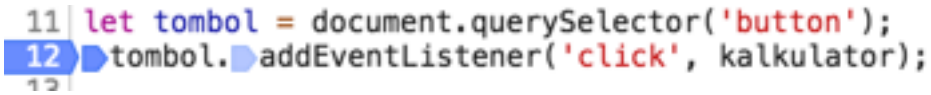
Atau **parseInt()** dilakukan sebelumnya:

```
let i1= parseInt(angka[0].value) ; //angka pertama
let i2= parseInt(angka[1].value) ; //angka kedua
angka[2].value= i1 + i2 ;
```

Catatan: *pesan.innerHTML* (obsolete) dapat diganti dengan *pesan.textContent*.

Breakpoint Baris per Baris

Breakpoint dapat dibuat melalui nomor baris code tersebut. Klik dan warna akan berubah menjadi biru.



Jika breakpoint tidak dibutuhkan lagi, maka tekan nomor baris tersebut dan warna biru kembali menjadi putih.

Pada saat break, maka tipe data dan nilai dari variable dapat dilihat (inspect).

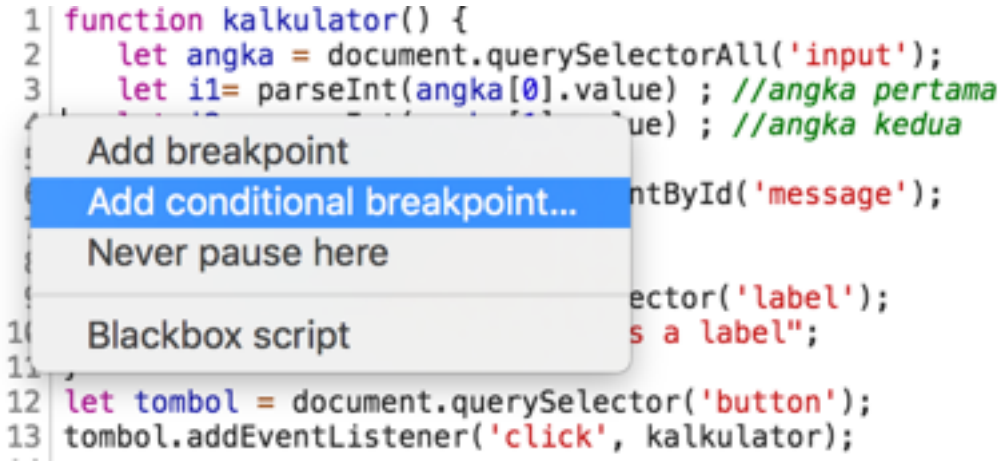
```
> typeof tombol
<"object"
> tombol
<  <button> = </button>
```

Breakpoint dapa juga disisipkan pada program dengan menggunakan instruksi "debugger".

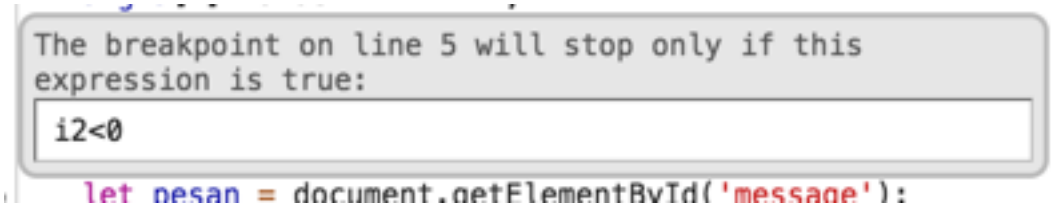
```
console.log('baris pertama');
console.log('baris kedua');
debugger;   // break disini
console.log('baris ketiga');
```

Breakpoint pada Kondisi tertentu

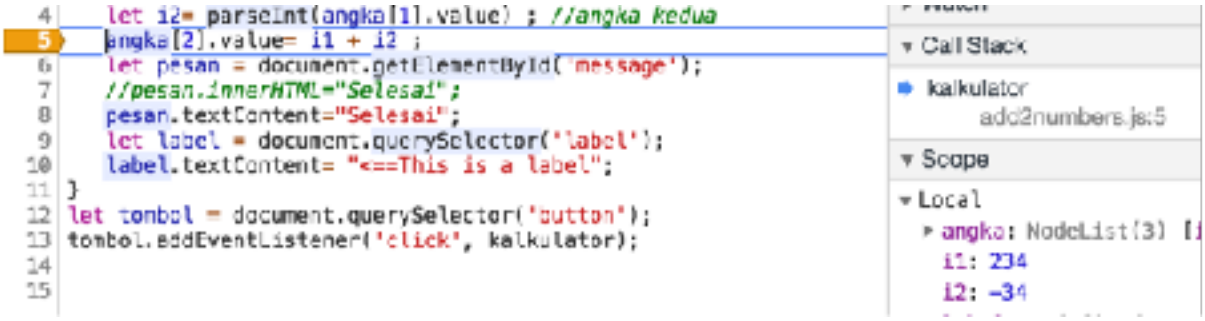
Program dapat dihentikan pada kondisi tertentu, misalnya mendapatkan angka negatif, atau kondisi tertentu terpenuhi seperti (i == 100), atau jika sesuatu menjadi true, atau toggling dari true menjadi false, dan sebaliknya.



Dari baris code yang akan ditentukan dengan kondisi, klik kanan. Pilih "Add conditional breakpoint", tentukan kondisi tersebut dan tekan Enter untuk mengaktifkan.



Pada contoh program akan berhenti bila nilai i2 negatif.



Nilai i2 adalah -34, dan oleh karenanya program berhenti pada baris no 5.