Programming Language Concepts

(HTML,CSS,Javascript)

* Christian Bartolomeus (2001538080) (http://barto12.blog.binusian.org/)
* Edward Suwandi (2001538963) (http://wardemon30.blog.binusian.org/)
* Kevin Anata Kristanto (2001536213) (http://kevinanata2001536213.blogspot.co.id/)
* Melvina Cynthia Liongga (2001537033) (http://melvinaclts.blog.binusian.org/)
* Ridwan Syarifudin (2001536743) (http://ridwan98.blog.binusian.org/)
1. **Syntax & Semantics**
2. **Syntax**

Syntax adalah adalah bentuk atau struktur dari ekspresi , pernyataan dan unit pemograman

HTML

* Elemennya tidak case sensitive

 contoh: <p> dan <P> artinya sama

* Ada tag start dan end

 <tagname> contents </tagname>

* Dalam elemen HTML dapat memuat elemen lainnya (nested elements). Biasanya seluruh dokumen HTML berbentuk nested.
* Terdapat juga elemen yang tidak memiliki content dan tidak perlu ditulis elemen end tag (empty element).

Contoh: <br> 🡪 untuk pindah baris



Contoh sintaks HTML:

 <!DOCTYPE html>

 <html lang="en">

 <head>

 <title>...</title>

 </head>

 <body>

 ...

 </body>

 </html>

CSS(Cascading Style Sheet)

* Selector🡪melakukan pemberian gaya pada *seluruh* elemen sebelum tanda {}

 contoh penulisan:

 P{……..} 🡪artinya memberikan gaya pada elemen P

* Property🡪menentukan parameter desain yang dapat diubah dari sebuah elemen yang dipilih oleh selector

 Contoh penulisan:

 P{color: …….;

 font size:….;

 }

 color🡪memberi warna pada elemen P

 font size🡪memberi ukuran font elemen P

* Value🡪  property yang ingin kita berikan tergantung tipe propertynya

contoh:

 P{color: #FFFF00 ;

 font size: 50px;

 }

 \* #FFFF00 kode warna putih sesuai format #RRGGBB

 \* ukuran font 50 pixel ( dikodekan dengan px)

Javascript

* Elemennya case sensitive
* Setiap statemen nya diakhiri dengan semicolon ;
* Untuk memasukkan program ini kedalam html dibutuhkan kode <script></script>

Contoh Javascript syntax :

<script>

var x = 5;

var y = 6;

var z = x + y;

document.getElementById("demo").innerHTML = z;

</script>

1. **Semantics**

HTML

Hanya dengan menulis kata – kata didalam body html maka saat di eksekusi kata – kata akan muncul

Javascripts

1. Evaluate MemberExpression.
2. Call GetValue(Result(1)).
3. Evaluate Expression.
4. Call GetValue(Result(3)).
5. Call ToObject(Result(2)).
6. Call ToString(Result(4)).

Return a value of type Reference whose base object is Result(5) and whose property name is Result(6).

1. **Names, Binding and Scopes**
2. **NAMES**

Names adalah string yang berisi character yang berguna untuk mengidentifikasi sebuah entity dalam program.

 **Names pada HTML:**

* Bersifat case-insensitive
* Character yang ada di dalam names tidak boleh mengandung tanda ‘<’ ‘/’ dan ‘>’ agar tidak terjadi misinterpretasi dengan tags
* Entity digunakan untuk menampilkan reserved character di HTML

Bentuk entity: 

Untuk menampilkan tanda ‘<’ agar tak terjadi misinterpretasi dengan tags,maka entity yang harus ditulis:  **&lt;** atau **&#60**

Berikut adalah contoh – contoh entity:

**Names pada CSS:**

* TIdak bersifat case sensitive
* Nama harus diawali dengan underscore ( \_ ),strip( - ) atau huruf (a – z) baru diikuti dengan tanda – tanda lainnya(biasanya bila dia wali ‘-‘ maka diikuti huruf atau ‘\_’. Names setidaknya sepanjang 2 karakter

Contoh: 

\*Names hanya terbaca di dalam proses komputer, apa yang ditampilkan terhadap user merupakan bahasa manusia sewajarnya.

**Names pada Javascript:**

* Variabelnya harus menggunakan nama – nama yang unik (identifiers)
* Nama variable bisa mengandung huruf,angka,underscore ataupun tanda dollar
* Bersifat case sensitive
* Keywords (mis: for, do, while, if dsb) tidak dapat dijadikan sebagai names
1. **BINDINGS**

 Dalam bahasa pemrograman binding merupakan cara untuk mengasosiasikan antara 2 atau lebih object/value item dalam satu scope. Hal ini dapat terjadi saat compiling (static link) ataupun saat runtime (dynamic link).

**Bindings dalam HTML:**

* Sebuah binding terdiri atas 2 item yaitu names dan value dan dipisahkan tanda kurung

Contoh:



* Sebuah elemen dapat terisi lebih dari 1 binding (multiple binding), setiap binding dipisahkan oleh tanda koma

Contoh:



* Value pada binding dapat berupa single value, variabel atau seluruh expression javascript yang valid
* Karena bindings berguna untuk menghubungkan satu objek/value ke lainnya, maka bindings di html sangat berguna untuk membuat berbagai macam fungsi dalam website. Contohnya: click binding, enable-disable binding, textinput binding dan contolflow binding(if,if-else,for,foreach)

**Bindings dalam CSS:**

* Bersifat menambah/mengurangi 1 class CSS yang diasosiasikan ke elemen DOM

Contoh dalam static class:

<div data-bind="css: { profitWarning: currentProfit() < 0 }">

   Profit Information

</div>

<script type="text/javascript">

    var viewModel = {

        currentProfit: ko.observable(150000) // Positive value, so initially we don't apply the "profitWarning" class

    };

    viewModel.currentProfit(-50); // Causes the "profitWarning" class to be applied

</script>

\*Hal ini berarti class profitWarning akan di-apply bila value currentProfit dibawah nilai 0 serta menghapus class tersebut

Contoh dalam dynamic class:

<div data-bind="css: profitStatus">

   Profit Information

</div>

<script type="text/javascript">

    var viewModel = {

        currentProfit: ko.observable(150000)

    };

    // Evalutes to a positive value, so initially we apply the "profitPositive" class

    viewModel.profitStatus = ko.pureComputed(function() {

        return this.currentProfit() < 0 ? "profitWarning" : "profitPositive";

    }, viewModel);

    // Causes the "profitPositive" class to be removed and "profitWarning" class to be added

    viewModel.currentProfit(-50);

</script>

\*Hal ini menyebabkan class profitPositive akan di-apply bila nilai currentProfit jika tidak maka secara otomatis akan di-apply class Profit

* Property isSevere memungkinkan untuk menggunakan beberapa CSS Class sekaligus

Contoh:

<div data-bind="css: { profitWarning: currentProfit() < 0, majorHighlight: isSevere }">

\*Menunjukan class profitWarning dan currentProfit digunakan sekaligus

**Bindings dalam Javascript:**

* Biasanya menggunakan framework (angularJS,EFL) untuk bindings antar obyek javascript, yakni memisahkan logika UI/UX dengan logika bisnis ( dalam sisi server)
* **ng-app**, biasanya ditempelkan ada di dalam tag awalHTML untuk menjadikan blok tersebut sebagai awal dari aplikasi AngularJS

**ng-model**, *directive*yang digunakan untuk mengambil data dari *field*di *form*

**{{ nama\_model }}**, mencetak data yang diambil dari **ng-model**

* Skema data binding:



\*data pada model dan view disinkronisasikan. Bila view berubah, model berubah dan berlaku sebaliknya

* Contoh:

\*pada tag body dipanggil angularJS via CDN, tag <html ng-app> berfungsi untuk menunjukan compiler bahwa aplikasi angularJS sedang digunakan dalam isi halaman. Ng-model adalah directive angularJS(menamai tiap variabel)

**3. SCOPES**

**Variabel Scope** (atau ruang lingkup variabel) adalah jangkauan kode program di mana perintah program masih bisa mengakses sebuah variabel. Variabel yang didefenisikan di dalam sebuah fungsi, secara **default** tidak dapat diakses oleh kode program di luar fungsi tersebut. Begitu juga sebaliknya, variabel yang didefenisikan di luar fungsi, tidak bisa diakses dari dalam fungsi. Scope dibedakan menjadi 2 yaitu lokal scope dan global scope. Variabel lokal hanya bisa diakses dalam 1 lingkup yang sama sedangkan variabel global dapat diakses di lingkup mana saja sebab

**Scope pada HTML:**

Dalam HTML scope artinya ruang lingkup untuk menyusun tabel dengan tag <td></td> dan <th></th>. Pengertian scope dalam HTML bukanlah pengertian lazim dari local scope maupun global scope sebab html adalah bahasa pemrograman yang menyusun ‘kerangka’ dari sebuah web. Dalam HTML tidak terjadi adanya inisiasi variabel – variabel tertentu, fungsi tersebut dilengkapi oleh CSS (styling web) dan javascript (logic control flow web).

**Scope pada CSS:**

Scope pada CSS berlaku local dan global dengan contoh sebagai berikut:



\*Pada potongan kode di atas,scope header memanggil variabel dan menghasilkan output warna putih, padahal di luar scope tersebut sudah dideklarasikan @var: red. Hal ini menunjukan apabila inisiasi global scope dan local scope dilakukan sekaligus,maka yang akan dieksekusi adalah perintah dalam local scope(scope paling bawah).

**Scope pada Javascript:**



\*potongan kode di atas merupakan gabungan dari local dan global scope. Engine javascript akan mencari variabel dalam scope lokalnya dulu, apabila tidak ditemukan variabel yang dimaksud dalam scope lokal maka engine akan mencari ke global scope.

1. **Data Types**

*\*Data types hanya tersedia dalam javascript dari bahasa – bahasa pemrograman web*

Tipe data dalam JavaScript dibedakan menjadi 2 kelompok, yakni **tipe data dasar** (*primitif*) dan **tipe data** **objek**.

Tipe data dasar terdiri dari tipe data **angka**, tipe data text (**string**), dan tipe data **boolean**. Tipe data **null** dan **undefined** juga merupakan tipe data dasar, namun memiliki jenis tersendiri.

Selain ke-5 tipe data dasar tersebut, tipe data lain yang ada di dalam **JavaScript** adalah **tipe data objek**. Contoh tipe data **objek** adalah tipe data tanggal (**date**), **array**, dan **fungsi**

**Tipe data primitive:**

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| String | represents sequence of characters e.g. "hello" |
| Number | represents numeric values e.g. 100 |
| Boolean | represents boolean value either false or true |
| Undefined | represents undefined value |
| Null | represents null i.e. no value at all |

Tipe data objek:

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| Object | represents instance through which we can access members |
| Array | represents group of similar values |
| RegExp | represents regular expression |

Tipe data yang ada dalam javascript bukanlah tipe data yang kita ketahui secara umum, tipe data angka ( tidak ada int atau float,semua angka otomatis disimpan sebagai float. Contoh deklarasi: ‘var a =12’ ), text (string), maupun Boolean akan berperilaku seolah – olah object. Tiap variabel yang memiliki tipe data tertentu tidak lepas dari fungsi yang melekat pada variabel tersebut. Hal ini akan dijelaskan lebih lanjut di bagian Object Oriented Programming.

1. **Expressions and Assignment Statements**

Ekspresi merupakan sarana dasar menentukan perhitungan dalam bahasa pemrograman.

♣ Untuk memahami evaluasi expressions, perlu akrab dengan perintah evaluasi operator dan operan.

♣ Essence bahasa imperatif adalah peran dominan dari assignment expressions.

**Expressions and Assignment Statements in Javascript**

JavaScript menggunakan operator = untuk menetapkan nilai ke variabel atau properti. Sebagai contoh:

i = 0 *// Set the variable i to 0.*

o.x = 1 *// Set the property x of object o to 1.*

Operator = mengharapkan operand kiri-sisinya menjadi lvalue: variabel atau objek properti (atau elemen array). Ia mengharapkan operand sisi kanan untuk menjadi arbitrary value dari jenis apa pun. Nilai assignment expression adalah nilai operand sisi kanan. Sebagai efek samping, operator = memberikan nilai di sebelah kanan untuk variabel atau properti di sebelah kiri sehingga referensi di masa depan dengan variabel atau properti mengevaluasi nilai.

Meskipun assignment expression biasanya cukup sederhana, Anda mungkin kadang-kadang melihat nilai dari sebuah assignment expression digunakan sebagai bagian dari ekspresi yang lebih besar. Misalnya, Anda dapat menetapkan dan menguji nilai dalam ekspresi yang sama dengan kode seperti ini:

(a = b) == 0

Jika Anda melakukan ini, pastikan Anda jelas pada perbedaan antara = dan == operator! Perhatikan bahwa = mendahului sangat rendah dan kurung biasanya diperlukan ketika nilai assignment yang akan digunakan dalam expression yang lebih besar.

Assignment operator memiliki kanan-ke-kiri associativity, yang berarti bahwa ketika beberapa assignment operator muncul dalam sebuah expression, mereka dievaluasi dari kanan ke kiri. Dengan demikian, Anda dapat menulis kode seperti ini untuk menetapkan nilai tunggal untuk beberapa variabel:

i = j = k = 0; *// Initialize 3 variables to 0*

1. **Control Structure Statement**

control structure adalah blok pemrograman yang menganalisis variabel dan memilih arah di mana untuk pergi didasarkan pada parameter yang diberikan. Rincian flow control jangka arah program mengambil (yang cara kontrol program "arus"). Oleh karena itu proses pengambilan keputusan dasar dalam komputasi; flow control menentukan bagaimana komputer akan merespon ketika mengingat kondisi dan parameter tertentu.

**Control structure statement di HTML dan CSS**

Memperkenalkan if-else conditional.

Jika Anda memiliki dua atau lebih program tindakan untuk memilih dari dalam program Anda, cara yang sempurna untuk menangani ini adalah melalui if-else conditional.

**if conditional.**

Bentuk paling sederhana dari control statement adalah if conditional, juga dikenal sebagai if statement atau if construct.

Sebuah if conditional mengevaluasi ekspresi ke nilai Benar atau Salah nya. Jika nilai sama dengan Benar, beberapa kode akan dieksekusi. Jika nilai False, kode tidak akan dieksekusi.

Berikut adalah bagaimana Anda membangun sebuah if conditional:

Mulailah dengan if.

Ikuti ini dengan condition expression dalam kurung, ().

Ikuti ini dengan kode yang akan dijalankan jika expression terpenuhi.

Kode ini harus dikelilingi oleh kurung kurawal.

Bentuk paling sederhana dari sebuah if conditional menggunakan variabel yang Anda harapkan mengandung nilai Benar atau Salah (dikenal sebagai variabel Boolean) sebagai ekspresi untuk dievaluasi oleh if statement:

$expression = true;

if($expression){

 print "true!";

}

Kode ini akan mencetak true !

Jika Anda ingin memeriksa $expression untuk melihat apakah itu sama dengan False, Anda bisa mendahului dengan tanda seru:.

$expression = true; if(!$expression){ print "false!"; ...

Control structure statement di Javascript

**Pernyataan ‘if’**

if( myVariable == 2 ) {

 myVariable = 1;

} else {

 myVariable = 0;

}

Jika myVariable telah 2 sekarang akan 1. Jika sudah apa pun selain 2 sekarang akan menjadi 0.

‘if’ statement juga dapat menguji untuk kejadian objek anak dari sebuah objek yang mungkin tidak ada. Sebagai contoh, beberapa browser memberikan document.body.style sementara beberapa browser lama bahkan tidak memberikan document.body. Dalam browser ini, menulis 'if’ (document.body.style)' hanya akan menghasilkan kesalahan (lihat bagian 'Variabel' subsection 'Avoiding errors with variables'). Untuk mengatasi masalah ini, kita bisa menulis ini:

if( document.body ) {

 if( document.body.style ) { etc. }

}

Namun, operator && memiliki fitur berguna yang bisa kita gunakan di sini untuk menggabungkan dua 'if' statement menjadi satu:

if( document.body && document.body.style ) { etc. }

Tes pertama akan false, sehingga browser tidak akan melanjutkan ke kedua. Hal ini dikenal sebagai arus pendek. Operator || memiliki fitur serupa, tetapi hanya akan mengevaluasi tes kedua jika yang pertama gagal.

JavaScript memahami bahwa jika '{' dan '}' (brace keriting) karakter yang ditinggalkan, maka hanya perintah berikutnya milik pernyataan tersebut:

if( x < 5 )

x++;

window.alert(x);

Di sini, peringatan akan selalu terjadi terlepas dari x, tapi x hanya akan bertambah jika x kurang dari 5. Hal ini mungkin tampak nyaman, karena memungkinkan Anda untuk membuat kode Anda sedikit kecil pendek, tapi saya sarankan menghindari sintaks ini. Itu membuat kode Anda sulit untuk dibaca, terutama jika Anda mulai nesting control structures anda. Hal ini juga memudahkan untuk lupa untuk menempatkan mereka di ketika Anda membutuhkan mereka, dan juga membuat kode debugging jauh lebih sulit, karena Anda akan perlu untuk kembali melalui kode Anda untuk menambahkan mereka sehingga Anda dapat menambahkan tes debugging ekstra. Cara terbaik adalah untuk selalu menggunakan curly braces, bahkan jika mereka adalah opsional.

Seperti biasa, ada pengecualian. Nested 'if' statements seperti ini bisa mulai mendapatkan sulit untuk mengelola:

if( myVariable == 2 ) {

 myVariable = 1;

} else {

 if( myVariable == 5 ) {

 myVariable = 3;

 } else {

 myVariable = 4;

 }

}

Dengan strategis menghapus curly braces, yang dapat berguna dikurangi untuk membangun ini (yang mungkin mengenali dari bahasa pemrograman lain) - mencatat bahwa else if' tidak ditulis sebagai 'elseif':

if( myVariable == 2 ) {

 myVariable = 1;

} else if( myVariable == 5 ) {

 myVariable = 3;

} else {

 myVariable = 4;

}

**'for' loop**

Ini adalah salah satu konstruksi yang paling umum digunakan. Biasanya, digunakan untuk siklus melalui isi sebuah array, atau untuk membuat jumlah tertentu objek baru, tetapi dapat melakukan banyak hal lebih berguna jika diperlukan. Sintaks dari 'untuk ‘for’ sebagai berikut:

for( starting\_initialise; continue\_as\_long\_as\_condition; do\_this\_each\_time )

Ini adalah di mana Anda mendefinisikan variabel baru yang akan Anda gunakan dalam loop, biasanya untuk digunakan dengan incremental counting. Seperti dengan semua variabel, Anda harus menyatakan mereka (jika Anda belum melakukannya sudah). Anda dapat mendefinisikan beberapa variabel jika diperlukan, menggunakan:

var myVariable1 = value, myVariable2 = another\_value;

Variabel-variabel ini tidak dibatasi untuk berada di dalam 'for'loop, dan akan tersedia untuk semua kode setelah loop (dalam lingkup yang sama sebagai loop).

continue\_as\_long\_as\_condition

Ini adalah di mana Anda menentukan conditons di mana loop harus terus mengeksekusi. sintaks ini persis sama seperti untuk 'if' statement, sehingga Anda dapat menerapkan lebih dari satu terus kondisi dengan menggunakan && atau || operator:

myVariable1 <= 5 && myVariable2 >= 70;

Jika kondisi ini tidak puas ketika untuk loop dimulai, maka tidak akan pernah loop melalui itu.

do\_this\_each\_time

Setelah akhir loop tercapai, ia akan melakukan apa pun yang Anda kirim ke sini. Biasanya, ini digunakan untuk increment or decrement a stepping variable, dan adalah mungkin untuk melakukan tindakan pada lebih dari satu variabel dengan memisahkannya dengan koma:

myVariable1++, myVariable2 -= 4

Berikut ini adalah contoh penuh.

for( var myVariable = 1; myVariable <= 5; myVariable++ ) {

 myArray[myVariable] = 1;

}

myArray[1] to myArray[5] are now 1.

 'while' loop

The 'while' loop identik dalam perilaku dengan 'for' loop, hanya tanpa pengaturan awal, dan loop-end tindakan. Ini akan terus berjalan selama kondisi satisfied:

var myVariable = 1;

while( myVariable <= 5 ) {

 myArray[myVariable] = 1;

 myVariable++;

}

myArray [1] to MyArray [5] sekarang 1.

Menggunakan fitur increment (dan decrement) Operator di sini, adalah mungkin untuk mempersingkat kode di dalam loop menjadi hanya 'myArray [myVariable ++] = 1;', dan itu akan memiliki efek yang sama persis. Pertama, itu akan menggunakan nilai myVariable indeks array cell, maka akan increment myVariable.

Hal ini juga bekerja secara terbalik; 'MyArray [++ myVariable] = 1;'. Pertama, itu akan increment nilai myVariable, maka akan menggunakan nilai baru ke indeks array cell . Jika saya telah melakukan ini, myArray [2] to MyArray [6] sekarang akan menjadi 1.

Fitur-fitur ini juga bekerja diluar loop, tetapi ini adalah di mana Anda akan paling sering melihat mereka, jadi saya telah menyertakan mereka di sini.

'**do - while' loop**

Hal ini mirip dengan while loop, tetapi dengan perbedaan penting. Kondisi ini dievaluasi pada akhir loop, yang berarti bahwa bahkan jika kondisi ini tidak pernah puas, masih akan berjalan melalui loop setidaknya sekali.

var myVariable = 1;

do {

 myArray[myVariable] = 1;

 myVariable++;

} while( myVariable <= 5 );

myArray [1] to MyArray [5] sekarang 1.

 'switch' statement

'Switch' statement seperti pengulangan ‘if’ statement, pengujian nilai tunggal untuk melihat apakah itu cocok dengan salah satu dari serangkaian nilai-nilai:

switch(myVar) {

 case 1:

 //if myVar is 1 this is executed

 case 'sample':

 //if myVar is 'sample' (or 1, see the next paragraph)

 //this is executed

 case false:

 //if myVar is false (or 1 or 'sample', see the next paragraph)

 //this is executed

 default:

 //if myVar does not satisfy any case, (or if it is

 //1 or 'sample' or false, see the next paragraph)

 //this is executed

}

Jika case is satisfied, kode luar kasus yang juga akan dieksekusi kecuali pernyataan break digunakan. Dalam contoh di atas, jika myVar adalah 1, kode untuk case 'sample', case false dan default semua akan dieksekusi juga. Solusinya adalah dengan menggunakan break; sebagai berikut (Penggunaan pernyataan break dijelaskan di bawah).

switch(myVar) {

 case 1:

 //if myVar is 1 this is executed

 break;

 case 'sample':

 //if myVar is 'sample' this is executed

 break;

 case false:

 //if myVar is false this is executed

 break;

 default:

 //if myVar does not satisfy any case, this is executed

 //break; is unnecessary here as there are no cases following this

}

**quick 'if' statement**

Ini dikenal sebagai conditional or ternary operator, dan merupakan cara mudah untuk menetapkan nilai yang berbeda untuk variabel, tergantung pada kondisi.

var myVariable = document.getElementById ? 1 : 0;

Ini identik dengan:

if( document.getElementById ) {

 var myVariable = 1;

} else {

 var myVariable = 0;

}

**break statement**

Menulis break di dalam switch, for,for-in, while or do - while control structure akan menyebabkan program untuk melompat ke akhir pernyataan tersebut. Jika Anda hanya menggunakan, misalnya:

for( var x = 1; x < 5; x++ ) {

 var y = 1;

 while( y < 7 ) {

 y++;

 if( y == 5 ) { break; }

 document.write(y);

 } }

script akan melompat melewati akhir while loop ketika y adalah 5. Tapi jika Anda menggunakan ini:

myForLoop:

for( var x = 1; x < 5; x++ ) {

 var y = 1;

 while( y < 7 ) {

 y++;

 if( y == 5 ) { break myForLoop; }

 document.write(y);

 }

}

script akan melompat melewati akhir untuk loop ketika y adalah 5.

**continue statement**

Menulis continue di dalam 'for', 'for-in', 'while' atau 'do-while'control structure akan menyebabkan program untuk melompat ke kondisi pengujian struktur dan re-mengevaluasi itu setelah dilakukan instruksi 'do\_this\_each\_time' . Jika Anda hanya menggunakan ini, misalnya:

for( var x = 1; x < 5; x++ ) {

 var y = 1;

 while( y < 7 ) {

 y++;

 if( y == 5 ) { continue; }

 document.write(y); } }

Script ini akan melompat ke kondisi uji while loop ketika y adalah 5 jadi 5 tidak akan pernah ditulis tapi 6 dan 7 akan. Jika Anda menggunakan ini sebagai gantinya:

myForLoop:

for( var x = 1; x < 5; x++ ) {

 var y = 1;

 while( y < 7 ) {

 y++;

 if( y == 5 ) { continue myForLoop; }

 document.write(y);

 } }

Di sini, script akan increment x sebagai bagian dari for loop dan kemudian kembali mengevaluasi for condition.

1. **Subprograms**

*\*Subprograms hanya ada dalam javascript dari ketiga bahasa pemrograman web*

Subprogram pada Javascript tidak mengenal procedure, tetapi hanya function dimana kasus procedure ditangani sebagai function tanpa nilai kembali (return value). Berikut contoh procedure :

<! -- js\_proc.html -->

<html>

<head>

<script language="javascript">

function myfunction()

{

    alert("STMIK-YMI Tegal")

}

</script>

</head>

<body>

<form>

<input type="button" onclick="myfunction()" value="Call Function">

</form>

</body>

</html>

    Output :
****

Procedure dengan parameter pass by value :

<! -- js\_proc2.html -->

<html>

<head>

<script language="javascript">

function myfunction(txt)

{

    alert("Hai "+txt)

}

</script>

</head>

<body>

<form>

Nama <input type="text" name="vstring">

<input type="button" onclick="myfunction('Hello')" value="Call Function">

</form>

</body>

</html>

****

Procedure dengan parameter pass by reference :
****

Fungsi :
Berikut ini contoh fungsi (function) dengan parameter pass by value dengan pemanggilan tak langsung atau melakukan assignment terhadap variable SUM terlebih dahulu.

<! -- js\_func1.html -->
<html>
<head>
<script language="javascript">
function total(a,b)
{
    return(a\*b)
}
</script>
</head>
<body>
<script language="javascript">
sum=total(2,3)
document.write(sum)
</script>
</body>
</html>

Output :
6

Berikut ini contoh fungsi (function) dengan parameter pass by reference dengan pemanggilan langsung.

<! -- js\_func2.html -->
<html>
<head>
<script language="javascript">
function totalx(a,b)
{
    return (a \* b)
}
</script>
</head>
<body>
<script language="javascript">
val1=window.prompt("Value 1 : ")
val2=window.prompt("Value 2 : ")
document.write(totalx(val1,val2))
</script>
</body>
</html>

1. **Abstract Data Type**

Untuk abstract data types kami bahas dalam javascript .

Didalam Bahasa pemograman C Abstract Data Types (ADT) lebih dikenal dengan data structure.

Tipe data abstrak pada JavaScript sangat berguna. tipe data abstrak meliputi object, array, dan fungsi.

Sebuah object merupakan kumpulan data dan fungsi yang bekerja sama untuk melakukan tugasnya. Salah satu fitur JavaScript yang paling penting adalah object-based approach. Semua tipe data abstrak, termasuk string, array, dan fungsi, adalah object. Pendekatan ini membuat objek yang sangat efisien dan mudah digunakan. Meskipun string dan array dianggap objek dalam JavaScript, perilaku mereka sedikit lebih khusus dari object umum lainnya

Array memfasilitasi penyimpanan data seperti dalam cara yang logis.

Fungsi adalah bagian dari kode yang dijalankan yang ditulis sekali tetapi disebut berulang kali dari berbagai program.

Kelebihan dari ADT:

* Memungkinkan untuk mengabaikan rincian pelaksanaan, jika struktur data dalam perubahan ADT kita hanya memodifikasi penggunaannya di satu tempat.
* Perubahan dan penambahan tidak mempengaruhi seluruh program, memperluas fungsionalitas dari sebuah ADT atau mengubah perilakunya tidak mempengaruhi bagian lain dari program tersebut.
* Membuat kode lebih mudah dibaca dan dijalankan.
* Program ini menjadi lebih terlihat benar.

Cara menginisiasi ADT pada javascript: $ npm install abstract-data-types

Individu tipe data abstrak yang dipakai dari fungsi awal yang ada sebagai sifat dari modul abstrak-jenis data. modul dapat diakses sebagai berikut.

var adt = require ('abstrak-jenis-data');

Fungsi ini dapat digunakan untuk membuat contoh dari tipe data abstrak seperti ini.

var q = adt.createQueue(); // untuk membuat antrian kosong

var ll = adt.createLinkedList(); // untuk membuat Linked List

var s = adt.createStack(); // untuk membuat Stack

var bt = adt.createBinaryTree(); // untuk membuat Binary Tree

var bst = adt.createBinarySearchTree(); // untuk Binary Search Tree

var g = adt.createGraph(); // untuk membuat Graph

var ht = adt.createHashTable(); // untuk membuat Hash Table

var h = adt.createHeap(); // untuk membuat Heap

ADT lebih banyak implementasinya pada PHP.

1. **Object Oriented Programming**

JavaScript adalah bahasa pemrograman yang paling penting dalam pengembangan website. JavaScript juga mendukung pemrograman berorientasi objek (*object oriented programming*), karena mengenal konsep **object**, **property**, dan **method**. Mungkin juga banyak yang melewatkan salah satu sifat dasar OOP yang terdapat pada JavaScript, yaitu **inheritance** atau pewarisan. Pewarisan dalam JavaScript disebut **prototype**.

Sifat – sifat OOP dalam javascript:

1. **Encapsulation :** Kemampuan untuk menyimpan terkait informasi, apakah data atau method secara bersama-sama dalam sebuah objek. Pemahaman lainnya adalah sebuah mekanisme untuk membungkus sebuah data maupun function sehingga menyembunyikan alur secara lengkapnya dan hanya menampilkan property tertentu yang dapat digunakan. Berikut contoh sederhana encapsulation.

|  |  |
| --- | --- |
| 12345678910111213141516171819202122232425262728 | var manusia = function () { //mulai membuat suatu object manusia    var namaLengkap = "Joni Esmon";   var reg = new RegExp(/\d+/);  //definisi lokal variabel dan pemberian nilai   return {    "setNamaLengkap" : function (newValue) {      if( reg.test(newValue) ) {        alert("Nilainya salah");      }      else {        namaLengkap = newValue;      }    }, //function ini digunakan untuk memberikan nilai pada lokal variabel namalengkap     "getNamaLengkap" : function () {     return namaLengkap;    } //function ini digunakan untuk menampilkan atau memanggil nilai pada lokal variabel namalengkap  }; }();   alert(manusia.getNamaLengkap());   // Joni Esmonmanusia.setNamaLengkap( "Jim Backlin" );alert(manusia.getNamaLengkap());  // Jim Backlinmanusia.setNamaLengkap( 42 ); // Nilainya salah; nama tidak berubah.manusia.namaLengkap = 42;     // tidak ada efek apapun karena tidak dapat merubah lokal variabel.alert(manusia.geNamaLengkap());  // Jim Backlin muncul kembali. |

1. **Aggregation :** Kemampuan menyimpan suatu objek di dalam objek lain

|  |  |
| --- | --- |
| 123456789101112131415161718192021 | var aggregate = function (objects) {    var map = {}; // map untuk menyimpan data    var a = [];   // array pemyimpan sementara     objects.forEach(function (d) {        var key = d.a > d.c ? String(d.c) + d.a : String(d.a) + d.c;        //digunakan untuk pemilihan key, jika a > c maka isi key = ca selainnya ac        map[key] = (map[key] || 0) + d.count;        //menyimpan kedalam variabel map berdasarkan key dan value    })    // misal, key === "bd"        for (var key in map) {        var k = key.toString().split("");        // membelah key menjadi array ["b", "d"]        a.push({ a: k[0], c: k[1], count: map[key] });        // menyimpan penyimpan array sementara sebagai appropriate object    }    return a;} var objects = [{a:'b',c:'d',count:1},{a:'b',c:'d',count:2},{a:'y',c:'d',count:4}];console.log(aggregate(objects)); |

1. **Inheritance :** Metode pewarisan dimana suatu class akan memiliki sifat yang serupa dengan class yang diturunkan

|  |  |
| --- | --- |
| 12345678910111213141516171819202122232425262728293031323334353637383940414243444546 | //======================= Definisi ClassA ========================var ClassA = function() {    this.name = "class A";}//membuat function utama dengan nama ClassA ClassA.prototype.print = function() {    console.log(this.name);} var a = new ClassA();//inisiasi function ClassA a.print();//menjalankan function print pada ClassA//======================= Definisi ClassB ========================var inheritsFrom = function (child, parent) {    child.prototype = Object.create(parent.prototype);};//function ini digunakan untuk menurunkan prototype atau sifat metode//yang terdapat pada kelas lebih tinggi var ClassB = function() {    this.name = "class B";    this.surname = "I'm the child";}//membuat function ClassB beserta nilai dari property inheritsFrom(ClassB, ClassA);//harus memanggil function diatas untuk mendapatkan sifat dari ClassA var b = new ClassB();//inisiasi variabel berdasarkan ClassB b.print();//menjalankan function print pada ClassB, function print tetap dapat dipanggil meski ClassB tidak membuat function print karena ClassB telah mendapatkan pewarisan sifat / property dari ClassA ClassB.prototype.print = function() {    ClassA.prototype.print.call(this); //memanggil function print yang terdapat ClassA    console.log(this.surname);}//melakukan override atau menambahkan sifat asli / property dari Class yang sudah diwarisi b.print(); //menjalankan function print setelah dilakuakn override |

1. **Polymorphism :** adalah kemampuan untuk membuat sebuah variabel, fungsi, atau sebuah benda yang memiliki lebih dari satu bentuk.
Penggunaan utama dari Polymorphism dalam Object-Oriented Programming adalah kemampuan objek yang milik berbagai jenis respon untuk menanggapi metode, bidang, atau memanggil properti dengan nama yang sama, masing-masing sesuai dengan perilaku tipe tertentu yang sesuai.

|  |  |
| --- | --- |
| 12345678910111213141516171819202122232425262728293031323334353637383940 | function Person(age, weight) {    this.age=age;    this.weight=weight;    this.getInfo=function() {      return "I am " + this.age + " years old " +        "and weighs " + this.weight +" kilo.";    }  }  //mmebuat function person   function Employee(age, weight, salary){    this.salary=salary;    this.age=age;    this.weight=weight;    this.getInfo=function() {      return "I am " + this.age + " years old " +        "and weighs " + this.weight +" kilo " +        "and earns " + this.salary + " dollar.";    }  }  //membuat function employee     Employee.prototype= new Person();  //function employee memiliki sifat yang menyerupai function Person  Employee.prototype.constructor=Employee;   function showInfo(obj) {    document.write(obj.getInfo()+"");  }//argumen 'obj' bisa berisi  apapun//function getInfo akan dijalankan bergantung dengan isi objek   var person = new Person(50,90);  //memberikan nilai pada function person (age, weight)  var employee = new Employee(43,80,50000);  //memberikan nilai pada function employee(age, weight, salary)     showInfo(person);  showInfo(employee);  //menampilan objek |

 Langkah membuat Object pada javascript :

1. Object() Constructor,  adalah fungsi yang menciptakan dan menginisialisasi sebuah objek. JavaScript menyediakan fungsi konstruktor khusus yang disebut Object () untuk membangun objek. return value dari Object () Constructor dimasukkan ke variabel.

|  |  |
| --- | --- |
| 123 | var book = new Object();   // Membuat object berdasarkan Object() Contrustorbook.subject = "Perl"; // Membuat properti dan memasukkan nilaibook.author  = "Mohtashim"; |

1. User-defined objects, Membuat object dengan mendefinisikan kedalam sebuah function

|  |  |
| --- | --- |
| 123456 | function book(title, author){            this.title = title;             this.author  = author; } //function untuk menjadikan objekvar myBook = new book("Perl", "Mohtashim"); //menggunakan dan mendefinisikan nilai  |

1. **Concurrency**

In computer science, concurrency is the decomposability property of a program, algorithm, or problem into order-independent or partially-ordered components or units.[1] This means that even if the concurrent units of the program, algorithm, or problem are executed out-of-order or in partial order, the final outcome will remain the same. This allows for parallel execution of the concurrent units, which can significantly improve overall speed of the execution in multi-processor and multi-core systems.

A number of mathematical models have been developed for general concurrent computation including Petri nets, process calculi, the Parallel Random Access Machine model, the Actor model and the Reo Coordination Language.

**Theory**

Concurrency theory has been an active field of research in theoretical computer science. One of the first proposals was Carl Adam Petri's seminal work on Petri Nets in the early 1960s. In the years since, a wide variety of formalisms have been developed for modeling and reasoning about concurrency.

**Models**

A number of formalisms for modeling and understanding concurrent systems have been developed, including:

 -The Parallel Random Access Machine

 -The Actor model

 -Computational bridging models such as the Bulk Synchronous Parallel (BSP) model

 -Petri nets

 -Process calculi

 -Communicating sequential processes (CSP) model

 -Tuple spaces, e.g., Linda

 -Simple Concurrent Object-Oriented Programming (SCOOP)

 -Reo Coordination Language

Some of these models of concurrency are primarily intended to support reasoning and specification, while others can be used through the entire development cycle, including design, implementation, proof, testing and simulation of concurrent systems. Some of these are based on message passing, while others have different mechanisms for concurrency.

The proliferation of different models of concurrency has motivated some researchers to develop ways to unify these different theoretical models. For example, Lee and Sangiovanni-Vincentelli have demonstrated that a so-called "tagged-signal" model can be used to provide a common framework for defining the denotational semantics of a variety of different models of concurrency, while Nielsen, Sassone, and Winskel have demonstrated that category theory can be used to provide a similar unified understanding of different models.

The Concurrency Representation Theorem in the Actor model provides a fairly general way to represent concurrent systems that are closed in the sense that they do not receive communications from outside. (Other concurrency systems, e.g., process calculi can be modeled in the Actor model using a two-phase commit protocol.) The mathematical denotation denoted by a closed system S is constructed increasingly better approximations from an initial behavior called ⊥S using a behavior approximating function progressionS to construct a denotation (meaning ) for S as follows:

 DenoteS ≡ ⊔i∈ω progressionSi(⊥S)

In this way, S can be mathematically characterized in terms of all its possible behaviors.

**Logics**

Various types of temporal logic can be used to help reason about concurrent systems. Some of these logics, such as linear temporal logic and computational tree logic, allow assertions to be made about the sequences of states that a concurrent system can pass through. Others, such as action computational tree logic, Hennessy-Milner logic, and Lamport's temporal logic of actions, build their assertions from sequences of actions (changes in state). The principal application of these logics is in writing specifications for concurrent systems.

**Practice**

Concurrent programming encompasses programming languages and algorithms used to implement concurrent systems. Concurrent programming is usually considered to be more general than parallel programming because it can involve arbitrary and dynamic patterns of communication and interaction, whereas parallel systems generally have a predefined and well-structured communications pattern. The base goals of concurrent programming include correctness, performance and robustness. Concurrent systems such as Operating systems and Database management systems are generally designed to operate indefinitely, including automatic recovery from failure, and not terminate unexpectedly (see Concurrency control). Some concurrent systems implement a form of transparent concurrency, in which concurrent computational entities may compete for and share a single resource, but the complexities of this competition and sharing are shielded from the programmer.

Because they use shared resources, concurrent systems in general require the inclusion of some kind of arbiter somewhere in their implementation (often in the underlying hardware), to control access to those resources. The use of arbiters introduces the possibility of indeterminacy in concurrent computation which has major implications for practice including correctness and performance. For example, arbitration introduces unbounded non-determinism which raises issues with model checking because it causes explosion in the state space and can even cause models to have an infinite number of states.

Some concurrent programming models include co-processes and deterministic concurrency. In these models, threads of control explicitly yield their time-slices, either to the system or to another process.

1. **Exception and Event Handling**
2. **Exception Handling**

Exception adalah singkatan dari Exceptional Events. Menurut definisi umum, exception adalah kondisi abnormal yang terjadi saat runtime. Runtime error atau kesalahan-kesalahan yang terjadi pada saat program berjalan diwujudkan dan exception. Exception dapat dibangkitkan secara otomatis oleh sistem Java runtime maupun sengaja kita buat melalui pernyataan tertentu untuk tujuan tertentu.

dapat kita simpulkan dari  pengertian diatas exception handling adalah suatu penanganan saat terjadi  kesalahan pada sebuah program akan terjadi suatu bentuk penanganan oleh java runtime ataupun dilakukan oleh kita yang disebut dengan exception handling dan exception handing sudah berjalan otomatis oleh java runtime atau dijalankan oleh kita melalui sebuah prosedur untuk mendapatkan suatu hasil tertentu atau dalam arti lain yaitu suatu cara untuk mengatasi kesalahan user

misalnya :
kita membuat program kalkulator , kita tau bahwa hanya type data number saja yang bisa di eksekusi dan kita tau ada kalanya user melakukan sebuah kesalahan yaitu dengan memasukan sebuah huruf , nah….. pada saat itu suatu program tanpa penangan yang tepat akan mengalami error dan program akan tertutup secara sendirinya .

nah sekarang sudah tahu fungsi  dari exception handling untuk mengatasi suatu kesalahan inputan user dengan suatu metode kita akan memberikan pesan pada saat user salah memasukan inputan dengan ini program tidak akan tertutup dengan sendirinya

Dalam javascript digunakan ‘try.. catch.. finally statements’ sebagai berikut:

<html>

 <head>

 <script type="text/javascript">

 <!--

 function myFunc()

 {

 var a = 100;

 try {

 alert("Value of variable a is : " + a );

 }

 catch ( e ) {

 alert("Error: " + e.description );

 }

 finally {

 alert("Finally block will always execute!" );

 }

 }

 //-->

 </script>

 </head>

 <body>

 <p>Click the following to see the result:</p>

 <form>

 <input type="button" value="Click Me" onclick="myFunc();" />

 </form>

 </body>

</html>

1. **Event Handling**

Event Handling adalah suatu metode untuk menangani sebuah event/aksi yang diberikan pengguna kepada suatu komponen GUI. Event adalah suatu peristiwa yang dipicu oleh pengguna pada suatu komponen, misalnya tombol ditekan. Dua paket yang biasa digunakan menangani kejadian adalah java.awt.event dan java.swing.event. Event handling dalam menangani event terbagi menjadi 3 macam yaitu Event Source, Event Listener, dan Event Handler.

1. .Event Source

Event Source adalah komponen yang mendapatkan suatu event yang kemudian ditangkap oleh Event Listener. Event Source dapat dibedakan dari penamaan komponen itu sendiri, misalnya tombolSimpan, tombolhapus, dan lainnya. Dengan penamaan Event Source ini, Sebuah Event Listener akan dapat mendeteksi dari mana Event Listener tersebut berasal.

1. .Event Listener

Event Listener digunakan untuk menangkap event yang terjadi pada komponen/Event Source.

1. .Event Handler

Event Handler berupa blok method dari listener yang menentukan proses selanjutnya setelah komponen mendapatkan event. Misalkan pada event tombol Simpan ditekan, event listener akan menangkap event pada Event Source tombol Simpan, kemudian Event Handler akan menyimpan data yang didefinisikan pada blok Event Handler

Contoh event handlers:

1. **Logic Programming Languages**

Logic Programming Languages akan dibahas dalam Javascript.

Logic programming Languages pada Javascript menggunakan LogicJS.

Logic programming lebih dikenal dengan language Prolog.

Logic Programming dipakai untuk membandingkan satu statement dengan statement yang lain;

Diinstall dengan npm

npm install logicjs

**Contoh :**

var or = logic.or,

 and = logic.and,

 eq = logic.eq,

 run = logic.run,

 lvar = logic.lvar,

 between = logic.between

//creates two unique logic variables

var x = lvar(),

 y = lvar()

//creates a 'goal'

g1 = or(

 and(eq(x,2), eq(y,3)),

 and(eq(x,y), eq(y,'dog'))

)

//runs goal asking for the possible values of x and y

run(g1, x) //[2, 'dog']

run(g1, y) //[3, 'dog']

run(g1, [x,y]) //[ [2, 3], ['dog', 'dog'] ]

//a goal is a sequence of assertions

//here, we assert that x is a value from 1 to 3

//and that y is either 1 or 2

g2 = and(

 between(1,3,x),

 or(eq(1,y),

 eq(2,y))

)

//get only the first 2 answers

run(g2, [x,y], 2) //[ [1, 1], [1, 2] ]

//get all answers

run(g2, [x,y]) //[ [1, 1], [1, 2], [2, 1], [2, 2], [3, 1], [3, 2] ]

1. **Functional Programming Languages**

Fungsi adalah salah satu bagian yang paling indah dari Javascript. Sebagai bahasa fungsional Javascript mengimplementasikan fungsi kelas pertama (first class function). Fungsi dapat disimpan dalam variabel, dikembalikan oleh fungsi lain, dan dikirimkan sebagai argumen untuk fungsi lainnya. Implementasi fungsi yang sangat fleksibel seperti ini membuka banyak kesempatan kepada pengembang untuk menuliskan kode yang bukan hanya berjalan dengan baik, tetapi juga sangat elegan dan indah.

Sebuah fungsi membungkus satu atau banyak perintah. Setiap kali kita memanggil fungsi, maka perintah-perintah yang ada di dalam fungsi tersebut dijalankan. Secara umum fungsi digunakan untuk penggunaan kembali kode (code reuse) dan penyimpanan informasi (information hiding). Implementasi fungsi kelas pertama juga memungkinkan kita menggunakan fungsi sebagai unit-unit yang dapat dikombinasikan, seperti layaknya sebuah lego. Dukungan terhadap pemrograman berorientasi objek juga berarti fungsi dapat kita gunakan untuk memberikan perilaku tertentu dari sebuah objek.

Sebuah fungsi pada Javascript dibuat dengan cara seperti berikut:

|  |  |
| --- | --- |
|   | **function** tambah(a, b) { hasil = a + b; **return** hasil;} |

Cara penulisan fungsi seperti ini dikenal dengan nama *function declaration*, atau deklarasi fungsi. Terdapat empat komponen yang membangun fungsi yang baru kita definisikan di atas, yaitu:

1. Kata kunci function, yang memberitahu Javascript bahwa kita akan membuat fungsi.
2. Nama fungsi, dalam contoh di atas adalah tambah. Dengan memberikan sebuah fungsi nama maka kita dapat merujuk ke fungsi tersebut dengan nama yang diberikan. Harus diingat bawa nama fungsi bersifat *opsional*, yang berarti **fungsi pada Javascript tidak harus diberi nama**. Kita akan membahas tentang hal ini lebih dalam nanti.
3. Daftar parameter fungsi, yaitu a, b pada contoh di atas. Daftar parameter ini selalu dikelilingi oleh tanda kurung (()). Parameter boleh kosong, tetapi tanda kurung wajib tetap dituliskan. Parameter fungsi akan secara otomatis didefinisikan menjadi variabel yang hanya bisa dipakai di dalam fungsi. Variabel pada parameter ini diisi dengan nilai yang dikirimkan kepada fungsi secara otomatis.
4. Sekumpulan perintah yang ada di dalam kurung kurawal ({}). Perintah-perintah ini dikenal dengan nama badan fungsi. Badan fungsi dieksekusi secara berurut ketika fungsi dijalankan.

Penulisan deklarasi fungsi (*function declaration*) seperti di atas merupakan cara penulisan fungsi yang umumnya kita gunakan pada bahasa pemrograman imperatif dan berorientasi objek. Tetapi selain deklarasi fungsi Javascript juga mendukung cara penulisan fungsi lain, yaitu dengan memanfaatkan ekspresi fungsi (*function expression*). Ekspresi fungsi merupakan cara pembuatan fungsi yang memperbolehkan kita melewatkan nama fungsi. Fungsi yang dibuat tanpa nama dikenal dengan sebutan fungsi anonim atau fungsi lambda. Berikut adalah cara membuat fungsi dengan ekspresi fungsi:

|  |  |  |
| --- | --- | --- |
|  |  | **var** tambah = **function** (a, b) { hasil = a + b; **return** hasil;}; |

Terdapat hanya sedikit perbedaan antara ekspresi fungsi dan deklarasi fungsi:

1. Penamaan fungsi. Pada deklarasi fungsi, kita langsung memberikan nama fungsi sesuai dengan sintaks yang disediakan Javascript. Menggunakan ekspresi fungsi kita pada dasarnya menyimpan sebuah fungsi anonim ke dalam variabel, dan nama fungsi adalah nama variabel yang kita buat. Perlu diingat juga bahwa pada dasarnya ekspresi fungsi *adalah* fungsi anonim. Penyimpanan ke dalam variabel hanya diperlukan karena kita akan memanggil fungsi nantinya.
2. Ekspresi fungsi dapat dipandang sebagai sebuah ekspresi atau perintah standar bagi Javascript, sama seperti ketika kita menuliskan kode var i = 0;. Deklarasi fungsi merupakan konstruksi khusus untuk membuat fungsi. Hal ini berarti pada akhir dari ekspresi fungsi kita harus menambahkan ;, sementara pada deklarasi fungsi hal tersbut tidak penting.

Karena pada Javascript sebuah fungsi juga adalah sekaligus sebuah objek, maka pada buku ini kita akan hampir selalu menggunakan ekspresi fungsi. Setiap kali kita menciptakan fungsi, pada dasarnya kita membuat sebuah objek Functionbaru, dengan nama yang kita berikan. Karenanya, secara eksplisit menuliskan bahwa kita membuat objek baru dan memperlakukan objek tersebut sama seperti perintah-perintah lain dalam program akan menyederhanakan kode program kita, yang pada akhirnya akan mempermudah kita mengerti kode kita nantinya.

Aturan pembuatan fungsi, baik ekspresi fungsi maupun deklarasi fungsi, sama dengan aturan penulisan ekspresi. Di mana kita dapat menuliskan ekspresi, kita dapat mendefinisikan fungsi juga. Karena aturan ini, maka kita juga dapat mendefinisikan fungsi di dalam fungsi lainnya. Fungsi yang berada di dalam fungsi lainnya memiliki akses terhadap semua variabel yang ada pada fungsi penampungnya. Keterhubungan fungsi di dalam fungsi ini dikenal dengan nama *closure*. Kita akan membahas tentang *closure* dan melihat bagaimana *closure* memberikan kemampuan ekspresi yang sangat besar kepada pengembang pada bagian berikutnya

**Sources:**

https://www.google.co.id/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwj8ntCzz53QAhXGvo8KHWkkDOkQFggaMAA&url=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FConcurrency\_(computer\_science)&usg=AFQjCNE5tHm01ua66b3MEQFFePENBmC8VQ

https://www.google.co.id/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwjMvqf62Z3QAhVMq48KHYcMBv4QFggaMAA&url=http%3A%2F%2Fgroups.engin.umd.umich.edu%2FCIS%2Fcourse.des%2Fcis400%2Fmaxim%2Flectures%2Fchp8.htm&usg=AFQjCNFo8X4sseYah\_wdzAMVe1RBVt8fsA&bvm=bv.138169073,d.c2I

<http://www2.southeastern.edu/Academics/Faculty/galkadi/401/notes/chapter7.doc>

<https://www.safaribooksonline.com/library/view/javascript-the-definitive/9781449393854/ch04s11.html>

<https://www.safaribooksonline.com/library/view/html-xhtml-and/9780470186275/ch27.html>

<http://www.howtocreate.co.uk/tutorials/javascript/controls>

http://keripix.github.io/belajar-js-dengan-contoh//posts/scope2.html

<http://books.gigatux.nl/mirror/javascriptprofessional/4233final/LiB0007.html>

<https://www.npmjs.com/package/abstract-data-types>

<https://github.com/mcsoto/LogicJS>

<http://stackoverflow.com/questions/448981/which-characters-are-valid-in-css-class-names-selectors>

<https://www.enlightenment.org/api/javascript>

[http://keripix.github.io/belajar-js-dengan-contoh//posts/scope.html](http://keripix.github.io/belajar-js-dengan-contoh/posts/scope.html)

<http://www.w3schools.com/tags/att_td_scope.asp>

<http://www.lucaongaro.eu/blog/2012/12/02/easy-two-way-data-binding-in-javascript/>

<http://knockoutjs.com/documentation/css-binding.html>

<http://knockoutjs.com/documentation/html-binding.html>

<http://lesscss.org/features/#features-overview-feature-scope>

<http://csswizardry.com/2013/05/scope-in-css/>

<https://bertzzie.com/knowledge/javascript/Fungsi-pada-Javascript.html>

http://www.duniailkom.com/tutorial-belajar-javascript-jenis-dan-pengertian-tipe-data-dalam-javascript/

<https://blog.sibudi.net/javascript-dan-object-oriented-programming-oop/>

https://reckys.wordpress.com/2016/03/30/mengenal-oop-object-oriented-programming-javascript/

<http://grup1ptistmiktegal.blogspot.co.id/2011/07/penggunaan-javascript-pada-html.html>

<http://searchsoa.techtarget.com/definition/data-type>

http://www.javatpoint.com/javascript-data-types