

Pemrograman Web II

Perhatian! Semua konten dalam ebook ini merupakan hasil copy paste dari website:
<https://www.duniailkom.com> dan Buku Pemrograman Web dengan PHP dan MySQL oleh Achmad Solichin, S.Kom

Duniailkom.com

DAFTAR ISI

1	Penggunaan Tools	7
1.1	Git dan Smartgit	7
1.2	Sublime.....	7
1.3	Composer	7
2	Dasar Pemrograman Php	8
2.1	Dasar Penulisan Kode Php.....	8
2.1.1	Penulisan Kode Php.....	8
2.1.2	Penulisan Komentar	10
2.2	Variabel dan Konstanta	13
2.2.1	Variabel	13
2.2.2	Konstanta	16
2.3	Tipe Data	19
2.3.1	Integer	20
2.3.2	Cara Penulisan Angka Hexadesimal Dalam PHP.....	21
2.3.3	Cara Penulisan Angka Oktal Dalam PHP.....	22
2.3.4	Cara Penulisan Angka Biner Dalam PHP	22
2.3.5	Float.....	23
2.3.6	String	24
2.3.7	Boolean.....	31
2.3.8	Array	33
2.3.9	Cara Penulisan Array dalam PHP	34
2.4	Operator Php.....	36
2.4.1	Jenis Operator Berdasarkan Jumlah Operand.....	36
2.4.2	Urutan Prioritas Operator dalam PHP.....	37
2.4.3	var_dump() dan print_r()	39
2.4.4	Operator Aritmatika	41

2.4.5	Operator String.....	42
2.4.6	Operator Logika.....	43
2.4.7	Operator Perbandingan.....	46
2.4.8	Operator Increment	49
2.4.9	Operator Assignment	50
2.4.10	Operator Bitwise	53
2.4.11	Operator Gabungan	57
2.5	Cara Mengubah Tipe Data PHP (Type Juggling dan Type Casting).....	58
2.5.1	Pengertian Type Juggling dalam PHP	58
2.5.2	Pengertian Type Casting dalam PHP	59
2.5.3	Aturan konversi data dalam PHP.....	60
3	Struktur Pemrograman	63
3.1	Logika IF.....	63
3.1.1	Aturan Penulisan Struktur IF dalam PHP.....	64
3.1.2	Alternatif Penulisan Struktur Logika IF.....	65
3.2	Logika Else	66
3.2.1	Aturan Penulisan Struktur IF-ELSE dalam PHP	66
3.2.2	Cara Penulisan Alternatif Struktur ELSE	68
3.3	Logika Else if	68
3.3.1	Aturan Penulisan Struktur ELSE-IF dalam PHP	69
3.3.2	Cara Penulisan Alternatif Struktur ELSE-IF	70
3.4	Logika Switch.....	71
3.4.1	Aturan Penulisan Struktur Switch dalam PHP	72
3.4.2	Perbedaan Antara struktur IF dengan Switch	75
3.5	Perulangan For	76
3.5.1	Cara Penulisan Struktur Perulangan For dalam PHP	77
3.5.2	Pengertian Infinity Loop	80

3.5.3	Pengertian Nested Loop (Perulangan Bersarang)	80
3.5.4	Cara Alternatif Penulisan Perulangan For dalam PHP	81
3.6	Perulangan do while	81
3.6.1	Cara Penulisan Struktur Perulangan While PHP	83
3.6.2	Penulisan Nested Loop untuk While	85
3.6.3	Cara Penulisan Alternatif Struktur Perulangan While	85
3.6.4	Pengertian Perulangan Do-While dalam PHP	86
3.6.5	Cara Penulisan Struktur Perulangan Do-While PHP	87
3.7	break dalam perulangan	88
3.7.1	Cara Penulisan Perintah Break	89
3.7.2	Cara Penggunaan Instruksi Break pada Perulangan Bersarang (Nested Loop)	90
3.7.3	Cara Penggunaan Continue pada Nested Loop	94
3.8	foreach	95
3.8.1	Cara Penulisan Perulangan Foreach dalam PHP	96
4	Fungsi	99
4.1	Pengertian	99
4.1.1	Cara Menggunakan Fungsi PHP	99
4.1.2	Pengertian Argumen dan Parameter dalam Fungsi PHP	99
4.1.3	Contoh Pemanggilan Fungsi PHP	100
4.2	Cara Penulisan Fungsi dan Pembuatannya	101
4.2.1	Contoh Pembuatan Fungsi PHP	101
4.2.2	Perbedaan antara Argumen dan Paramater	103
4.3	Variabel Scope, Global, dan static	103
4.3.1	Contoh Variabel Scope dalam Fungsi PHP	104
4.3.2	Pengertian Global Variabel	105
4.3.3	Pengertian Static Variabel	106
4.4	Pengecekan tipe data argumen	106

4.4.1	Pentingnya Pengecekan Tipe Data Argumen	106
4.4.2	Cara Pengecekan Tipe Data Argumen Fungsi.....	108
4.5	Parameter Default.....	110
4.5.1	Cara Penulisan Default Parameter dalam PHP.....	110
4.5.2	Penempatan Default Parameter	112
4.6	Variabel Parameter	113
4.6.1	Cara Pembuatan Fungsi dengan Variable Parameter	113
4.7	Fungsi bawaan Php.....	116
4.7.1	Case Conversion	116
4.7.2	Mengubah Huruf Kecil ke Besar	117
4.7.3	Mengubah Huruf Pertama Awal String	117
4.7.4	Mengubah Huruf Pertama Setiap Kata	118
4.7.5	Substr.....	119
4.7.6	Cara Mengambil Karakter Dari Awal String.....	120
4.7.7	Cara Mengambil Karakter Dari Akhir String	121
4.7.8	trim	122
4.7.9	Menambahkan Karakter yang akan Dihapus (character_mask)	124
4.7.10	explode	125
4.7.11	strpos.....	129
4.7.12	Pengertian Fungsi strpos().....	132
4.7.13	implode.....	132
5	Koneksi Database Mysql	135
5.1	Jenis-jenis koneksi ke Mysql.....	135
5.1.1	Koneksi MySQL dengan mysql extension	135
5.1.2	Koneksi MySQL dengan mysqli extension	136
5.1.3	Koneksi MySQL dengan PDO (PHP Data Objects)	136
5.1.4	Perbandingan antara mysqli, PDO dan mysql extension	136

5.1.5	Cara Penulisan mysql extension, mysqli extension, dan PDO (PHP Data Objects)	137
5.2	Mengaktifkan PDO	138
5.2.1	Jenis Database Server yang didukung PDO	139
5.3	Membuat koneksi Mysql PDO	142
5.3.1	Cara Menampilkan Pesan Kesalahan (error) pada PDO	143
5.4	Menampilkan Data PDO	144
5.4.1	Mengubah Cara Menampilkan Data PDO	145
5.5	Menginput data mysql PDO	146
5.6	Menggunakan Prepared Statement	148
5.6.1	Pengertian Prepared Statements MySQL.....	148
6	Form Php.....	152
6.1	Membuat dan Memproses Form HTML dengan PHP	152
6.1.1	Mengenal Struktur Dasar Form HTML (atribut action dan method)	152
6.1.2	Cara Mengirimkan Nilai Form HTML ke dalam PHP	154
6.1.3	Cara Menampilkan nilai Form HTML dengan PHP (\$_GET dan \$_POST)	156
6.2	Metode GET dan POST	158
6.2.1	Keunggulan dan Kelemahan Method Form 'GET'	158
6.2.2	Keunggulan dan Kelemahan Method Form 'POST'	160
6.3	Variabel GET POST dan REQUEST.....	160
6.3.1	Perbedaan variabel global \$_GET, \$_POST dan \$_REQUEST	161
6.4	Register Global dan Register Long Array.....	162
6.4.1	Cara Mengakses Nilai Form dengan Register Global	163
6.4.2	Cara Mengakses Nilai Form dengan Register Long Array	166
6.5	Validasi form php	167
6.5.1	Pentingnya Melakukan Validasi Nilai Form	167
6.5.2	Memeriksa Ketersediaan Variabel Form dengan Fungsi isset()	169
6.5.3	Memeriksa Apakah Variabel Form Telah Diisi.....	170

6.5.4	Menyeleksi Tipe Data Objek Form	172
6.6	Mencegah XSS dan HTML injection.....	173
6.6.1	Contoh Cross-site Scripting dan HTML injection.....	174
6.6.2	Cara Mencegah Cross-site Scripting dan HTML injection	175
6.6.3	Validasi dengan htmlspecialchars atau strip_tags?	178
6.6.4	Mengenal Fungsi htmlentities()	178
6.7	Mengirim variabel antar halaman php.....	179
6.7.1	Redirect halaman dengan fungsi header(Location:)	180
6.7.2	Cara Menampilkan Pesan Kesalahan Form	180
7	File dan Directory	183
7.1	Penanganan File	183
7.2	Membuat dan Menghapus Direktori.....	189
7.2.1	Membuat Direktori Baru	189
7.2.2	Menghapus Direktori.....	190
7.3	Manipulasi File	192
7.4	UploadFile	194
8	Session dan Cookies.....	197
8.1	Cookies.....	0
9	OOP Review.....	1
9.1	Enkapsulasi	1
9.2	Variabel \$this.....	1
9.3	property dan method dalam class	1
9.4	autoload	2

1 PENGGUNAAN TOOLS

1.1 GIT DAN SMARTGIT

1.2 SUBLIME

1.3 COMPOSER

2.1 DASAR PENULISAN KODE PHP

2.1.1 PENULISAN KODE PHP

Seperti bahasa pemrograman lainnya, PHP juga memiliki aturan penulisan seperti case sensitivity (perbedaan antara huruf besar dan kecil), cara mengakhiri sebuah baris perintah, dan pengaruh menggunakan spasi dalam membuat kode program PHP. Berikut adalah aturan dasar penulisan kode PHP:

1. Case Sensitivity (perbedaan huruf besar dan kecil) dalam PHP

PHP tidak membedakan huruf besar dan kecil (case insensitive) untuk penamaan fungsi (function), nama class, maupun keyword bawaan PHP seperti echo, while, dan class. Ketiga baris berikut akan dianggap sama dalam PHP:

```
<?php
Echo "Hello World";
ECHO "Hello World";
EcHo "Hello World";
?>
```

Akan tetapi, PHP membedakan huruf besar dan huruf kecil (case sensitive) untuk penamaan variabel, sehingga \$nama, \$Nama dan \$NAMA akan dianggap sebagai 3 variabel yang berbeda. Sering kali error terjadi dikarenakan salah menuliskan nama variabel, yang seharusnya menggunakan huruf kecil, ditulis dengan huruf besar.

```
<?php
$andi="Andi";
echo $Andi; // Notice: Undefined variable: Andi
?>
```

Untuk mengatasi perbedaan ini, disarankan menggunakan huruf kecil untuk seluruh kode PHP, termasuk variabel, fungsi maupun class. Jika membutuhkan nama variabel yang terdiri dari 2 kata, karakter spasi bisa digantikan dengan underscore (_)

2. Penulisan Baris Perintah dalam PHP

Statement (baris perintah) di dalam PHP adalah kumpulan perintah PHP yang menginstruksikan PHP untuk melakukan sesuatu. Baris perintah ini bisa terdiri dari satu baris singkat (seperti perintah echo untuk menampilkan text di layar) atau bisa sesuatu yang lebih rumit dan terdiri dari beberapa baris, seperti kondisi if, atau kode perulangan (loop).

Berikut adalah contoh beberapa baris perintah dalam PHP:

```
<?php
echo "Hello, world";
sebuah_fungsi(21, "duniailkom");
$a = 1;
$nama = "duniailkom";
$b = $a / 25.0;
if ($y == $z) {
echo "Tampilkan Tabel";
}
?>
```

Terlihat dari beberapa contoh baris perintah diatas, PHP menggunakan tanda semicolon (titik koma) ";" sebagai tanda akhir baris perintah.

Kumpulan baris perintah yang menggunakan tanda kurung kurawal seperti kondisi IF atau perulangan (loop) tidak membutuhkan tanda titik koma setelah kurung penutup.

```
<?php
if (true) {
echo "Perintah dijalankan"; // tanda titik koma harus ditulis
} // tidak diperlukan tanda titik koma setelah tanda kurung kurawal
?>
```

3. Karakter Spasi dan Tab dalam PHP

Secara umum, karakter spasi dan tab diabaikan di dalam eksekusi program PHP. Anda boleh memecah sebuah statement menjadi beberapa baris, atau menyatukan beberapa statement dalam sebuah baris yang panjang. Seperti contoh berikut:

```
<?php
echo "Ini kalimat pertama"; echo "Ini kalimat kedua";
$nama="duniailkom";
?>
```

Baris perintah itu sama artinya dengan

```
<?php
echo "Ini kalimat pertama";
echo "Ini kalimat kedua";
$nama = "duniailkom";
?>
```

Walaupun contoh pertama lebih menghemat tempat, namun saya lebih menyarankan contoh kedua, dimana kita mengusahakan agar setiap statement berada pada satu baris saja, dan menambahkan beberapa spasi di awal untuk memudahkan membaca kode program.

Keuntungan penghematan beberapa baris dan beberapa byte dari sebuah file PHP tidak akan sebanding dengan efek sakit kepala yang anda dapati sewaktu mencoba memahami kode program yang dibuat 3 bulan kedepan (karena sering menggabungkan beberapa statement dalam satu baris). Menambahkan komentar pada bagian kode yang lebih rumit sebagai penjelasan juga sangat disarankan.

2.1.2 PENULISAN KOMENTAR

Komentar di perlukan untuk memberikan penjelasan kepada orang lain yang membaca kode kita. Komentar sepenuhnya akan diabaikan oleh PHP pada saat eksekusi. Walaupun anda berfikir bahwa mungkin hanya anda yang akan membaca kode program tersebut, namun itu adalah ide yang bagus. Saya sendiri sering bingung ketika memahami beberapa baris kode program setelah beberapa bulan tidak melihatnya.

Komentar yang baik adalah komentar singkat dan tidak terlalu panjang, namun memberikan penjelasan tentang kegunaan kode atau variabel tersebut dibuat.

Selain sebagai tempat membuat penjelasan, sifat komentar yang tidak akan dieksekusi oleh PHP, memberikan keuntungan lain dalam proses pembuatan program. Saya sering menjadikan beberapa baris kode program sebagai komentar ketika mencari tau penyebab error kode program yang sedang dibuat (proses debugging).

PHP menyediakan beberapa cara untuk membuat komentar, dan semuanya berasal dari bahasa pemrograman populer lain seperti C, C++, dan Unix Shell.

1. Metode Komentor Unix Shell

Disebut sebagai metode komentar Unix Shell, karena cara memberikan komentar ini berasal dari sistem Unix. Metode ini menggunakan karakter tanda pagar atau hash mark (#). PHP akan mengabaikan seluruh text yang terdapat setelah tanda pagar sampai akhir baris atau tag penutup PHP (mana yang terlebih dahulu didapati).

Karena sifatnya yang hanya mempengaruhi satu baris saja, Metode Komentor Unix Shell efektif digunakan untuk membuat komentar pendek.

```
<?php
$nilai = $p * exp($r * $t);
?>
```

Beberapa programmer juga sering menggunakan karakter # untuk memisahkan bagian kode PHP dengan bagian lainnya, seperti berikut:

```
#####  
## Falidasi Form Register  
#####  
... Kode program PHP disini
```

Ketika anda membuat kode program PHP dan HTML yang saling berkaitan, Komentor Unix Shell ini bisa digunakan seperti contoh berikut:

```
<?php $nama = "duniailkom"; # Set $nama menjadi duniailkom ?>  
<br> <?php echo $nama; ?>  
... kode HTML berikutnya
```

2. Metode Komentor C++

Metode komentar ini meminjam cara membuat komentar dari bahasa pemrograman C++. Hampir sama dengan metode komentar Unix Shell, metode komentar C++ ini berlaku hanya untuk sebuah baris atau sampai tag penutup PHP, Tetapi kali ini karakter yang digunakan adalah dua kali garis miring (two slashes), yakni “//”.

Karena sifatnya yang sama seperti Unix Shell, semua contoh tanda ‘#’ dapat diganti dengan ‘//’, berikut contohnya:

```
$nilai = $p * exp($r * $t); ///////////////////////////////////  
// Falidasi Form Register  
////////////////////////////////////  
... Kode program PHP disini  
<?php $nama = "duniailkom"; // Set $nama menjadi duniailkom. ?>  
<br> <?php echo $nama; ?>  
... kode HTML berikutnya
```

Saya lebih menyukai menggunakan Metode Komentor C++ ini dibandingkan dengan metode Unix Shell, karena tombolnya mudah dicapai. Untuk menekan tanda “#”, anda harus menekan tombol shift di keyboard terlebih dahulu, namun untuk mengetik tanda “//” kita tinggal menggunakan satu tangan kanan untuk mencapainya di keyboard.

3. Metode Komentar C

Jika metode komentar Unix Shell dan C++ efektif untuk membuat komentar pendek, untuk membuat komentar yang panjang, PHP meminjamnya dari bahasa C. Metode komentar ini disebut juga tipe komentar blok karena sifatnya yang harus diberikan tanda tutup untuk mengakhiri komentar.

Untuk memulai komentar, kita menuliskan sebuah garis miring dan diikuti dengan tanda bintang (/*). Semua text setelah tanda tersebut akan dianggap sebagai komentar sampai PHP menemukan tanda tutup, yakni karakter bintang dan diikuti dengan garis miring (*). Metode komentar C ini dapat mencakup beberapa baris.

Berikut adalah contoh penggunaan Metode Komentar C

```
<?php
/* Dalam bagian ini kita akan membuat
beberapa variabel dan memberikan nilai awal.
Nilai awal ini hanya sebagai contoh saja, jadi jangan dianggap serius
*/
$nama = "Andi";
$a = 10;
$situs = "duniailkom";
$b= 2014;
?>
```

Metode Komentar C ini juga berguna untuk "mengomentari" beberapa baris program agar tidak dijalankan oleh PHP, Seperti contoh berikut:

```
<?php
$a= 3;
/*
bagian ini tidak akan dijalankan oleh PHP
$b = 7;
$c = 8;
*/
?>
```

Namun anda perlu berhati-hati untuk tidak membuat blok komentar yang saling bertumpuk, seperti kode berikut:

```
<?php
$a = 12;
/*
$j = 10; /* Ini adalah komentar */
$k = 11;
Ini adalah komentar
*/
?>
```

Dalam contoh diatas, PHP akan gagal menjalankan kode program dan menghasilkan error di sebabkan komentar yang saling berhimpitan (overlapping).

2.2 VARIABEL DAN KONSTANTA

2.2.1 VARIABEL

Dalam pemrograman, variabel adalah suatu lokasi penyimpanan (di dalam memori komputer) yang berisikan nilai atau informasi yang nilainya tidak diketahui maupun telah diketahui (wikipedia).

Dalam definisi bebasnya, variabel adalah kode program yang digunakan untuk menampung nilai tertentu. Nilai yang disimpan di dalam variabel selanjutnya dapat dipindahkan ke dalam database, atau ditampilkan kembali ke pengguna.

Nilai dari variabel dapat di isi dengan informasi yang diinginkan dan dapat dirubah nilainya pada saat kode program sedang berjalan. Sebuah variabel memiliki nama yang digunakan untuk mengakses nilai dari variabel itu. Jika anda memiliki pengetahuan dasar tentang bahasa pemrograman, tentunya tidak asing dengan istilah variabel.

Sama seperti variabel dalam bahasa pemrograman lainnya, variabel dalam PHP digunakan untuk menampung nilai inputan dari user, atau nilai yang kita definisikan sendiri. Namun PHP memiliki beberapa aturan tentang cara penggunaan dan penulisan variabel.

2.2.1.1 ATURAN PENULISAN VARIABEL DALAM PHP

1. Penulisan variabel harus diawali dengan tanda \$

Variabel di dalam PHP harus diawali dengan dollar sign atau tanda dollar (\$).

Setelah tanda \$, sebuah variabel PHP harus diikuti dengan karakter pertama berupa huruf atau underscore (_), kemudian untuk karakter kedua dan seterusnya bisa menggunakan huruf, angka atau underscore (_). Dengan aturan tersebut, variabel di dalam PHP tidak bisa diawali dengan angka.

Minimal panjang variabel adalah 1 karakter setelah tanda \$.

Berikut adalah contoh penulisan variabel yang benar dalam PHP:

```
<?php
$i;
$nama;
$Umur;
$_lokasi_memori;
$ANGKA_MAKSIMUM;
?>
```

Dan berikut adalah contoh penulisan variabel yang salah:

```
<?php
$4ever; //variabel tidak boleh diawali dengan angka
$_salah satu; //varibel tidak boleh mengandung spasi
$nama*^; //variabel tidak boleh mengandung karakter khusus: * dan ^
?>
```

2. Variabel dalam PHP bersifat case sensitif

PHP membedakan variabel yang ditulis dengan huruf besar dan kecil (bersifat case sensitif) , sehingga \$belajar tidak sama dengan \$Belajar dan \$BELAJAR, ketiganya akan dianggap sebagai variabel yang berbeda.

Untuk menghindari kesalahan program yang dikarenakan salah merujuk variabel, disarankan menggunakan huruf kecil untuk seluruh nama variabel.

```
<?php
$andi="Andi";
echo $Andi; // Notice: Undefined variable: Andi
?>
```

Dalam contoh diatas, PHP mengeluarkan error karena tidak menemukan variabel \$Andi.

3. Cara Memberikan Nilai kepada Variabel

Sama seperti sebagian besar bahasa pemrograman lainnya, untuk memberikan nilai kepada sebuah variabel, PHP menggunakan tanda sama dengan (=). Operator 'sama dengan' ini dikenal dengan istilah Assignment Operators.

Perintah pemberian nilai kepada sebuah variabel disebut dengan assignment. Jika variabel tersebut belum pernah digunakan, dan langsung diberikan nilai awal, maka disebut juga dengan proses inisialisasi.

Berikut contoh cara memberikan nilai awal (inisialisasi) kepada variabel:

```
<?php
$nama = "andi";
$umur = 17;
$pesan = "Saya sedang belajar PHP di duniailkom.com";
?>
```

4. Variabel dalam PHP tidak memerlukan deklarasi terlebih dahulu

Jika anda pernah mempelajari bahasa pemrograman desktop seperti Pascal, C, C++, dan Visual Basic, di dalam bahasa pemrograman tersebut, sebuah variabel harus dideklarasikan terlebih dahulu sebelum digunakan.

Namun di dalam PHP, variabel tidak perlu deklarasikan terlebih dahulu. Anda bebas membuat variabel baru di tengah-tengah kode program, dan langsung menggunakannya tanpa di deklarasikan terlebih dahulu.

```
<?php
$andi="Andi";
echo $andi;
?>
```

PHP memiliki keyword `var` untuk mendefinisikan variable, keyword ini digunakan untuk PHP versi 4 kebawah. PHP versi 5 tidak membutuhkan keyword ini, dan penggunaannya akan menghasilkan error, seperti contoh berikut ini:

```
<?php
// kode program dibawah ini akan menghasilkan error
// Parse error: syntax error, unexpected 'var' (T_VAR)
var $andi="Andi";
echo $andi;
?>
```

5. Variabel dalam PHP tidak bertipe

Dalam kelompok bahasa pemrograman, PHP termasuk Loosely Type Language, yaitu jenis bahasa pemrograman yang variabelnya tidak terikat pada sebuah tipe tertentu.

Hal ini berbeda jika dibandingkan dengan bahasa pemrograman desktop seperti Pascal atau C, dimana jika anda membuat sebuah variabel bertipe integer, maka variabel itu hanya bisa menampung nilai angka, dan anda tidak akan bisa mengisinya dengan huruf.

Di dalam PHP, setiap variabel bebas diisi dengan nilai apa saja, seperti contoh berikut:

```
<?php
$a = 17; // nilai variabel a berisi angka (integer)
$a = "aku"; // nilai variabel a diubah menjadi kata (string)
$a = 17.42; // nilai variabel a diubah menjadi desimal (float)
?>
```

6. Variabel Sistem PHP (Predefined Variables)

Predefined Variables atau terjemahan bebasnya Variabel Sistem PHP, adalah beberapa variabel yang telah didefinisikan secara sistem oleh PHP, dan kita sebaiknya tidak membuat variabel dengan nama yang sama.

Beberapa contoh Predefined Variables dalam PHP adalah:

```
$GLOBALS , $_SERVER , $_GET , $_POST , $_FILES , $_COOKIE , $_SESSION  
, $_REQUEST , $_ENV, $php_errormsg, $HTTP_RAW_POST_DATA,  
$http_response_header, $argc, $argv, $this.
```

Daftar list Predefined Variables tersebut saya ambil dari manual PHP di <http://www.php.net/reserved.variables>, di dalam manual tersebut juga dijelaskan bahwa mungkin masih terdapat beberapa variabel sistem PHP selain list diatas, hal ini tergantung dengan jenis web server, versi PHP yang digunakan, dan beberapa faktor lainnya. Namun kebanyakan variabel sistem PHP menggunakan tanda \$_ pada awal nama variabel, namun tidak selalu.

2.2.1.2 CARA MENAMPILKAN NILAI VARIABEL

Untuk menampilkan nilai atau isi dari variabel, kita tinggal menampilkannya dengan perintah echo atau print, seperti berikut ini:

```
<?php  
    $a='Saya Sedang belajar PHP';  
    $b=5;  
  
    print $a;  
    echo $b;  
?>
```

Hasil yang didapat adalah:

```
Saya Sedang belajar PHP5
```

Perhatikan bahwa kedua nilai variabel ditampilkan tanpa spasi diantaranya. Hal ini terjadi karena di dalam program PHP saya tidak menyisipkan spasi untuk pemisah diantara kedua variabel.

Walaupun kita akan membahasnya lebih lengkap pada tutorial mengenai string, kita juga bisa menampilkan variabel langsung di dalam string jika string tersebut berada di antara tanda kutip dua ("):

```
<?php  
    $a=5;  
    $b="Sedang belajar PHP $a";  
  
    echo $b;  
    // hasil: Saya Sedang belajar PHP 5  
?>
```

2.2.2 KONSTANTA

Dalam bahasa pemrograman, Konstanta (constant) adalah suatu lokasi penyimpanan (dalam memory) yang berisikan nilai yang sifatnya tetap dan tidak bisa diubah sepanjang program berjalan (wikipedia).

Berbeda dengan variabel yang isi/nilainya dapat diubah bahkan dihapus selama program berjalan, sebuah konstanta jika telah diberikan nilai, tidak dapat diubah lagi dalam kode program. Hal ini sesuai dengan namanya, yakni konstant.

2.2.2.1 ATURAN PENULISAN KONSTANTA PHP

1. Cara Pendefinisikan Konstanta dalam PHP

Jika variabel di dalam PHP dibuat dengan menambahkan tanda dollar, seperti: \$nama. Untuk membuat konstanta PHP menyediakan 2 cara:

- a) Menggunakan kata kunci (keyword) const.
- b) Menggunakan fungsi define.

Untuk mendefinisikan konstanta dengan kata kunci const, caranya mirip dengan menambahkan nilai kepada sebuah variabel, namun didahului kata const. Berikut adalah contoh penulisannya:

```
<?php
const situs = "www.duniailkom.com";
echo situs; // www.duniailkom.com
?>
```

Jika menggunakan fungsi define, fungsi ini membutuhkan 2 nilai, yakni nama konstanta, dan nilainya. Seperti contoh berikut ini:

```
<?php
define("situs", "www.duniailkom.com");
echo situs; // www.duniailkom.com
?>
```

Aturan penamaan konstanta sama seperti variabel, yakni untuk karakter pertama hanya boleh menggunakan huruf dan underscore (_), dan untuk huruf kedua dan seterusnya, boleh menggunakan huruf, angka dan underscore. Sehingga sebuah konstanta juga tidak boleh diawal angka atau mengandung karakter khusus seperti #, *, atau &.

Sedikit catatan tentang perbedaan pendefinisian konstanta menggunakan kata kunci const dan fungsi define:

Pembuatan konstanta dengan keyword const hanya dapat digunakan pada top-level scope, yakni harus dalam lingkungan global PHP. Sehingga kita tidak bisa menggunakan const di dalam function, loop, atau kondisi if.

Apabila anda membutuhkan konstanta di dalam fungsi, maka harus menggunakan keyword define.

2. Konstanta PHP bersifat Case Sensitif

Sama seperti variabel, konstanta dalam PHP bersifat case sensitif, sehingga perbedaan huruf besar dan kecil dianggap berbeda. GAJI, GaJi, dan gaji merupakan 3 konstanta yang berbeda.

Walaupun kita boleh menggunakan huruf kecil dalam penulisan konstanta, kesepakatan programmer PHP menganjurkan menggunakan HURUF BESAR untuk penulisan konstanta. Tujuannya agar lebih mudah untuk membedakan dengan variabel (dimana variabel dianjurkan menggunakan huruf kecil).

3. Nilai Konstanta PHP Tidak Dapat Diubah

Jika sebuah konstanta telah didefinisikan, kita tidak bisa merubah nilai tersebut.

Contoh error konstanta:

```
<?
define("GAJI", 5000000);
echo GAJI; echo "<br />";
define("GAJI", 50000);
?>
```

Jika kode program tersebut dijalankan, berikut tampilannya:

```
5000000
Notice: Constant GAJI already defined in
D:\xampp\htdocs\belajar\test.php on line 4
```

4. Konstanta hanya dapat berisi tipe data tertentu

Konstanta dalam PHP hanya dapat berisi tipe data sederhana (disebut juga jenis tipe skalar), yakni: boolean, integer, float dan string. Hal ini berbeda dengan variabel, yang dapat juga berisi tipe data turunan seperti array, objek atau resources.

```
<?php
define("GAJI", 5000000);
echo GAJI; echo "<br />";
define("GAJI_PEGAWAI", array( 1000000,1500000));
?>
```

Hasil contoh diatas akan menghasilkan error sebagai berikut:

```
5000000
Warning: Constants may only evaluate to scalar values in
D:\xampp\htdocs\belajar\test.php on line 4
```

Di dalam contoh diatas saya mencoba memberikan nilai array sebagai nilai dari konstanta GAJI_PEGAWAI, namun PHP mengeluarkan error yang menjelaskan bahwa konstanta hanya dapat berisi

nilai dengan tipe skalar saja. Tipe skalar ini adalah boolean, integer, float dan string. Kita akan mempelajari tipe-tipe data ini dalam tutorial selanjutnya.

5. Konstanta Sistem PHP (Predefined Constant)

Sama seperti variabel, PHP juga telah membuat beberapa konstanta yang telah didefinisikan dan tidak bisa di ubah nilainya. Namun karena banyaknya modul yang dapat ditambahkan kedalam PHP, Predefined Constant dalam PHP akan bertambah tergantung modul yang ada. Namun sebagai contoh, berikut adalah Predefined Constant dalam sistem inti PHP:

```
PHP_VERSION, PHP_MAJOR_VERSION, PHP_MINOR_VERSION,
PHP_RELEASE_VERSION, PHP_VERSION_ID, PHP_EXTRA_VERSION, PHP_ZTS,
PHP_DEBUG, PHP_MAXPATHLEN, PHP_OS, PHP_SAPI, PHP_EOL, PHP_INT_MAX,
PHP_INT_SIZE, DEFAULT_INCLUDE_PATH, PEAR_INSTALL_DIR,
PEAR_EXTENSION_DIR, PHP_EXTENSION_DIR, PHP_PREFIX, PHP_BINDIR,
PHP_BINARY, PHP_MANDIR, PHP_LIBDIR, PHP_DATADIR, __LINE__, __FILE__,
__DIR__, __FUNCTION__, __CLASS__, __TRAIT__, __METHOD__,
__NAMESPACE__,
```

Daftar list Predefined Constant diatas saya ambil langsung dari manual PHP di <http://php.net/manual/en/reserved.constants.php>. Kebanyakan dari konstanta tersebut menyimpan nilai yang dapat membantu kita dalam membuat program PHP, khususnya untuk debugging, saya akan membahasnya pada lain kesempatan.

Dalam aplikasi dunia nyata, penggunaan konstanta tidak akan sesering penggunaan variabel, namun jika anda membutuhkan sebuah variabel yang tidak bisa ditimpa nilainya, konstanta merupakan pilihan yang tepat.

Dalam tutorial Belajar PHP Dasar selanjutnya, kita akan masuk kedalam tipe data. Untuk pembahasan pertama, kita akan membahas tentang tipe data integer di dalam PHP.

2.3 TIPE DATA

Sebuah variabel atau konstanta merupakan 'tempat' dari data. Di dalam bahasa pemrograman (dan juga PHP), data yang diinput kedalam variabel atau konstanta akan memiliki tipe tertentu. Tipe-tipe ini nantinya menentukan bagaimana cara kita memprosesnya. Beberapa tipe data terdengar familiar, seperti tipe data angka, desimal dan text. Namun kita juga akan menemukan tipe data lain seperti boolean dan array.

Untuk tipe data pertama yang akan dibahas adalah tipe data angka bulat, atau disebut dengan tipe data Integer.

2.3.1 INTEGER

Tipe data integer adalah tipe data yang berupa angka bulat seperti: 1, 22, dan -172. Tipe data integer umum digunakan untuk data dengan angka bulat, seperti harga barang, jumlah stock dan jumlah mahasiswa. Jika data yang kita miliki kemungkinan akan mengandung pecahan, maka tipe data yang digunakan adalah float (akan dibahas dalam tutorial berikutnya).

Nilai integer dapat bernilai positif (+) maupun negatif (-). Jika tidak diberi tanda, maka diasumsikan nilai tersebut adalah positif.

Berikut contoh penulisan bilangan integer dalam PHP:

```
<?php
$umur=21;
$harga=15000;
$rugi=-500000;
echo $umur; //21
echo "<br />";
echo $harga; //15000
echo "<br />";
echo $rugi; //-500000
?>
```

Untuk variabel dengan angka integer, kita bisa melakukan operasi matematis seperti penambahan, pengurangan, pembagian dan lain-lain, seperti contoh berikut ini:

```
<?php
$a=14;
$b=16;
$c= $a + $b;
echo $c; // 30
$d=$a * $b;
echo $d; // 224
?>
```

Karena PHP tidak memerlukan pendeklarasian variabel, maka ketika sebuah variabel berisi angka bulat, maka secara otomatis variabel tersebut di sebut sebagai variabel integer.

Jangkauan angka integer bergantung kepada kemampuan komputasi komputer, namun biasanya dimulai dari -2,147,483,648 sampai +2,147,483,647, atau 32bit. Jika terdapat menungkinan angka yang dihasilkan dari kode program kita berada diluar jangkauan ini, sebaiknya menggunakan tipe data float.

Secara teknis, jangkauan angka integer ini sama dengan jangkauan tipe data LONG pada bahasa C. Namun dikarenakan bahasa C tidak memberikan spesifikasi khusus seberapa besar jangkauan LONG, anda mungkin mendapat hasil yang berbeda.

Untuk mengetahui nilai maksimal tipe data integer pada komputer, PHP menyediakan konstanta PHP_INT_MAX. Berikut adalah hasil nilai PHP_INT_MAX yang saya jalankan:

```
<?php
print PHP_INT_MAX; // 2147483647
?>
```

Selain digunakan untuk menampung angka dengan base 10 (disebut juga angka desimal), tipe data integer digunakan juga untuk menampung angka base 16 (hexadesimal), base 8 (octal), dan base 2 (binary).

Jika anda baru mempelajari bahasa pemrograman, anda boleh melewati penjelasan tentang integer base 16 (hexadesimal), base 8 (octal), dan base 2 (binary) yang akan dijelaskan dibawah ini. Karena tipe data ini tidak terlalu sering digunakan, dan mungkin akan membuat bingung jika belum pernah mempelajarinya.

2.3.2 CARA PENULISAN ANGKA HEKSADESIMAL DALAM PHP

Angka heksadesimal (atau hexadecimal) adalah angka khusus yang bilangannya terdiri dari 16 digit, yaitu angka 0-9, dan huruf A-F. Angka heksadesimal ini biasanya digunakan untuk pemrosesan yang berkaitan dengan perhitungan komputer.

Dalam pemrograman web kita akan menemukan angka ini pada penulisan kombinasi warna pada CSS. Sebagai contoh, warna merah ditulis: ff0000, biru: 0000ff, abu-abu: cccccc, dan kombinasi warna lainnya dalam CSS menggunakan angka heksadesimal.

Untuk membuat sebuah variabel berisi angka heksadesimal, kita menulis huruf "0x" (angka 0 dan huruf 'x') sebelum angka yang ingin diinput. Karakter "0x" menginstruksikan kepada PHP bahwa angka setelahnya adalah heksadesimal. Misalkan, angka heksadesimal 54FA ditulis dengan 0x54FA.

Berikut contoh penulisan bilangan integer heksadesimal dalam PHP:

```
<?php
$angka_desimal= 31;
$angka_heksadesimal=0x1F; //1F heksadesimal = 31 desimal
echo $angka_desimal; //31
echo "<br />";
echo $angka_heksadesimal; //31
?>
```

Pada contoh diatas, variabel \$angka_desimal dan \$angka_heksadesimal sama-sama ditampilkan dengan nilai 31 (PHP secara tidak langsung mengkonversi nilai \$angka_heksadesimal menjadi nilai desimal).

2.3.3 CARA PENULISAN ANGKA OKTAL DALAM PHP

Bilangan oktal adalah bilangan yang terdiri dari 8 digit, yaitu karakter 0-7. Bilangan oktal tidak terlalu sering digunakan. Untuk menuliskan bilangan oktal ke dalam variabel PHP, kita menggunakan tanda "0" (angka nol) diawal angka. Angka desimal 511 ditulis dalam bentuk oktal 777, sehingga penulisannya dalam PHP menjadi 0777. Berikut contoh penulisan bilangan integer oktal dalam PHP:

```
<?php
$angka_desimal= 511;
$angka_oktal=0777; //777 oktal = 511 desimal
echo $angka_desimal; //511
echo "<br />";
echo $angka_oktal; //511
?>
```

Jika anda perhatikan, penulisan angka dengan 0 didepan sebuah angka sering kita tulis dalam fungsi matematika atau catatan sehari-hari. Namun, karena fungsinya sebagai penanda bilangan oktal, sedapat mungkin hindari kebiasaan menambahkan angka 0 di depan nilai desimal di dalam PHP. Jika yang dimaksud adalah supaya tampilan angka menjadi cantik, misalkan untuk pengurutan nomor : 01, 02, 03.. dst. PHP menyediakan fungsi khusus untuk keperluan ini.

2.3.4 CARA PENULISAN ANGKA BINER DALAM PHP

Bilangan biner (atau binary) adalah bilangan yang terdiri dari 2 digit saja, yaitu 0 dan 1. Prinsip bilangan biner inilah yang mendasari perhitungan komputer. Bilangan biner kadang disebut juga dengan bilangan logika, yakni logika benar (ditandai dengan angka 1), dan logika salah (ditandai dengan angka 0).

Di dalam PHP, angka integer biner ditulis dengan awalan 0b (angka nol, dan huruf b). Nilai desimal 222, dalam notasi biner biner ditulis 11011110, sehingga untuk menyimpannya dalam variabel PHP ditulis menjadi 0b11011110.

Berikut contoh penulisan bilangan integer biner dalam PHP:

```
<?php
$angka_desimal= 222;
$angka_biner=0b11011110; //11011110 biner = 222 desimal
echo $angka_desimal; //222
echo "<br />";
echo $angka_biner; //222
?>
```

Tipe data integer adalah tipe data penting dalam pemrograman. Dalam tutorial kali ini kita telah mempelajari cara penulisan bilangan integer dalam PHP dengan penulisan untuk karakter desimal, heksadesimal, oktal, dan biner.

2.3.5 FLOAT

Tipe data float (disebut juga tipe data floating point, atau real number) adalah tipe data angka yang memiliki bagian desimal di akhir angka, atau memiliki floating point (floating point adalah istilah dalam bahasa Inggris untuk menyebut tanda “titik” yang menandakan bilangan desimal). Contoh angka float adalah seperti: 0,9 atau 3,14.

Tipe data float cocok digunakan untuk variabel yang akan berisi angka pecahan, seperti nilai IPK, hasil pembagian, atau hasil komputasi numerik yang angkanya tidak bisa ditampung oleh data integer.

Sama seperti tipe data integer, jangkauan angka float bergantung kepada komputasi prosesor yang digunakan, walaupun pada umumnya berupa angka mulai dari 1.7×10^{-308} sampai dengan 1.7^{+308} dengan 15 digit keakuratan. Anda tidak perlu khawatir dengan limit angka float ini, karena selain untuk membuat aplikasi matematis tingkat tinggi, kita tidak akan menggunakan angka float sampai 15 digit.

Dikarenakan perbedaan cara penulisan bilangan float di Eropa dan Amerika dengan Indonesia (sama dengan bahasa pemrograman lain pada umumnya), didalam PHP penulisan nilai desimal ditandai dengan tanda “titik”, bukan “koma” seperti yang biasa kita gunakan sehari-hari. Nilai 0,87 harus ditulis menjadi 0.87. PHP akan menampilkan pesan error jika sebuah nilai ditulis dengan angka 0,87.

Cara Penulisan Tipe Data Float dalam PHP

PHP mendukung 2 cara penulisan tipe data float, yang pertama yaitu penulisan desimal sehari-hari, seperti 0.17 atau 9.47 dan yang kedua berupa penulisan format scientific notation, seperti 0.314E1, atau 12.0E-3.

Penggunaan scientific notation digunakan untuk menyederhanakan penulisan, 0.314E1 adalah sama dengan 0.314×10^1 atau 3.14, dan 12.0E-3 sama dengan 12.0×10^{-3} , atau 0.012.

Berikut contoh penulisan bilangan float dalam PHP:

```
<?php
$angka_float1= 0.78;
$angka_float2= 14.99;
$angka_scientific1=0.314E1;
$angka_scientific2=0.3365E-3;
echo $angka_float1; // 0.78
echo "<br />";
echo $angka_float2; //14.99
echo "<br />";
echo $angka_scientific1; //3.14
echo "<br />";
echo $angka_scientific2; //0.0003365
```

```
?>
```

Sama seperti tipe data integer, variabel dengan tipe data float juga dapat melakukan operasi numerik seperti penambahan, pembagian, perkalian, dan lain-lain. Berikut adalah contoh operasi matematis dengan tipe data float:

```
<?php
    $a=10.66;
    $b=12.4;
    $c= $a + $b;
    echo $c; // 23.06
    $d=$a / $b;
    echo $d; // 0.85967741935484
?>
```

Penjelasan lanjutan tentang operasi matematis, akan kita bahas dalam tutorial khusus tentang operator matematis dalam PHP.

Dalam tutorial belajar PHP ini, kita telah mempelajari tipe data float, atau disebut juga tipe data floating point atau real number. Selanjutnya kita akan membahas tentang Tipe Data String dan Cara Penulisan String dalam PHP.

2.3.6 STRING

Tipe data string adalah tipe data yang berisi text, kalimat, atau kumpulan karakter. Sebagai contoh, “a”, “saya sedang belajar PHP” atau “tUT0r1al pHp?!” semuanya adalah string.

Tipe data string mungkin adalah tipe data yang paling sering digunakan, dan memiliki banyak fitur yang disediakan PHP. Karakter yang didukung saat ini adalah 256 karakter ASCII. List karakter ASCII tersebut dapat dilihat di <http://www.ascii-code.com>.

2.3.6.1 CARA PENULISAN TIPE DATA STRING DALAM PHP

PHP menyediakan 4 cara penulisan tipe data string, yakni Single Quoted, Double Quoted, Heredoc, dan Nowdoc. Kita akan mempelajarinya lebih dalam dalam tutorial ini.

2.3.6.2 PENULISAN TIPE DATA STRING DENGAN SINGLE QUOTED

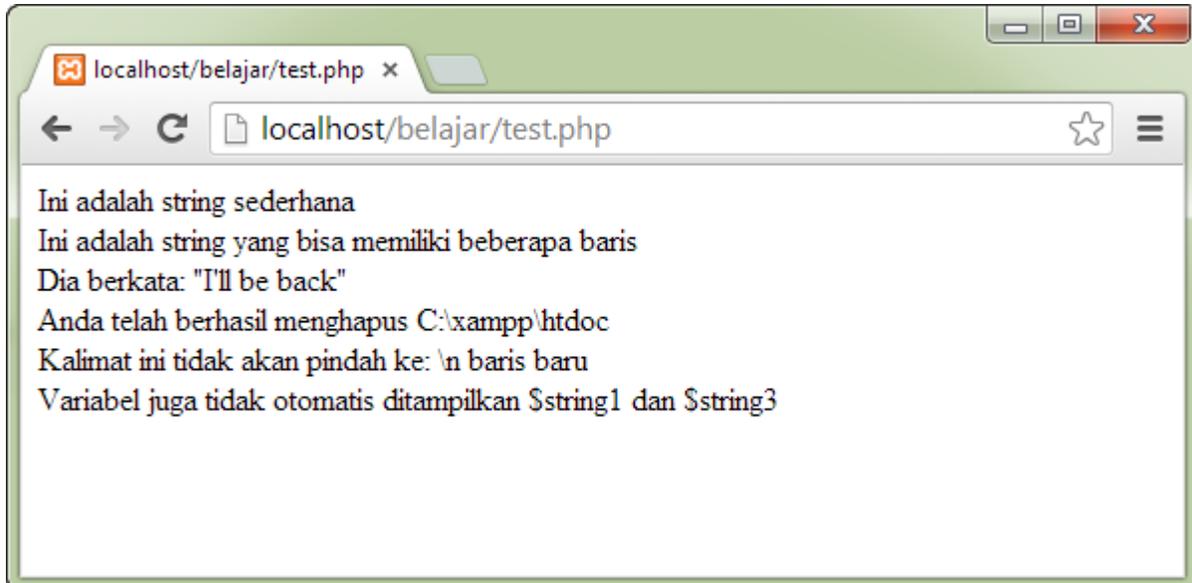
Penulisan tipe data string menggunakan single quoted atau tanda petik satu (karakter ‘) merupakan cara penulisan string yang paling sederhana. Kita tinggal membuat sebuah kata atau kalimat, dan menambahkan tanda petik satu di awal dan akhir kalimat.

Untuk string yang didalamnya juga terdapat tanda petik satu, kita harus mendahuluinya dengan karakter backslash (\) agar tidak dianggap sebagai penutup string. Dan jika di dalam string anda ingin menulis tanda backslash, kita harus menuliskannya dengan 2 kali (\\).

Berikut adalah contoh penulisan tipe data string menggunakan metode single quoted:

```
<?php
$string1='Ini adalah string sederhana';
$string2='Ini adalah string
yang bisa memiliki beberapa
baris';
$string3='Dia berkata: "I\'ll be back"';
$string4='Anda telah berhasil menghapus C:\\xampp\\htdocs';
$string5='Kalimat ini tidak akan pindah ke: \n baris baru';
$string6='Variabel juga tidak otomatis ditampilkan $string1 dan
$string3';
echo $string1; echo "<br>";
echo $string2; echo "<br>";
echo $string3; echo "<br>";
echo $string4; echo "<br>";
echo $string5; echo "<br>";
echo $string6;
?>
```

Jika contoh tersebut dijalankan, berikut tampilannya di browser:



Contoh Penulisan Tipe Data String Dengan metode single quoted. Pada contoh diatas, saya membuat beberapa karakter khusus seperti ", \n, dan variabel yang dimulai dengan tanda dollar (\$). Ketiga karakter khusus ini ditampilkan secara karakter aslinya ke dalam browser.

2.3.6.3 PENULISAN TIPE DATA STRING DENGAN DOUBLE QUOTED

Cara kedua dalam penulisan tipe data string dalam PHP adalah dengan menggunakan Double Quoted atau tanda petik dua (karakter "). Walaupun seperti tidak ada perbedaan dengan menggunakan single quote, hasil yang di dapat akan sangat berbeda.

Dengan double quoted, PHP akan memproses karakter-karakter khusus seperti carriage return (\n), dan karakter tab (\t) dan juga memproses setiap variabel (yang ditandai dengan tanda \$ didepan kata).

Di karenakan metode double quoted melakukan pemrosesan terlebih dahulu, maka untuk menampilkan karakter khusus seperti tanda petik (karakter '), tanda dollar (karakter \$) dan tanda-tanda khusus lainnya, kita harus menggunakan backslash (karakter \). Berikut adalah tabel karakter khusus untuk double quoted string:

Cara Penulisan String	Karakter Yang Ditampilkan
\"	Karakter Tanda petik dua
\n	Karakter Newline
\r	Karakter Carriage return
\t	Karakter Tab
\\	Karakter Backslash
\\$	Karakter Dollar Sign
\{	Karakter Pembuka Kurung Kurawal
\}	Karakter Penutup Kurung Kurawal
\[Karakter Pembuka Kurung Siku
\]	Karakter Penutup Kurung Kurawal
\0 sampai \777	Karakter ASCII menggunakan nilai oktal
\x0 sampai \xFF	Karakter ASCII menggunakan nilai heksadesimal

Sebagai contoh penggunaan double quoted string, saya akan menggunakan contoh yang sama dengan single quoted string, agar dapat dilihat perbedaannya:

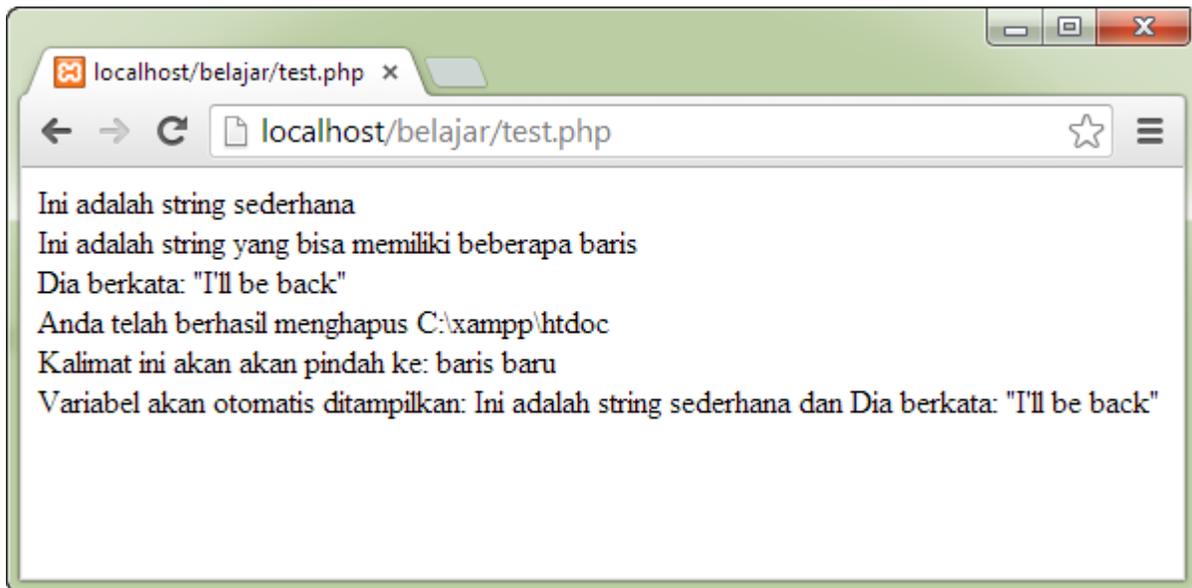
```
<?php
$string1="Ini adalah string sederhana";
$string2="Ini adalah string
yang bisa memiliki beberapa
```

```

baris";
$string3="Dia berkata: \"I'll be back\"";
$string4="Anda telah berhasil menghapus C:\\xampp\\htdocs";
$string5="Kalimat ini akan akan pindah ke: \n baris baru";
$string6="Variabel akan otomatis ditampilkan: $string1 dan $string3";
echo $string1; echo "<br \>";
echo $string2; echo "<br \>";
echo $string3; echo "<br \>";
echo $string4; echo "<br \>";
echo $string5; echo "<br \>";
echo $string6;
?>

```

Dan hasil kode PHP tersebut adalah:



Perhatikan perbedaannya pada hasil \$string3, \$string5 dan \$string6.

Pada \$string3, kita harus mem-blackslash tanda petik dua karena itu merupakan karakter khusus dalam double quoted string. Pada \$string5, tanda \n yang merupakan karakter khusus untuk baris baru, tapi karena kita menampilkannya di browser, karakter ini tidak akan terlihat, tetapi jika kita menulis hasil string ini kedalam sebuah file text, kalimat tersebut akan terdiri dari 2 baris.

Pada \$string6, terlihat bahwa string dengan petik dua akan memproses variabel \$string1 dan \$string3 sehingga tampil hasilnya di web browser. Fitur ini akan sangat bermanfaat jika kita sering menampilkan variabel didalam sebuah string.

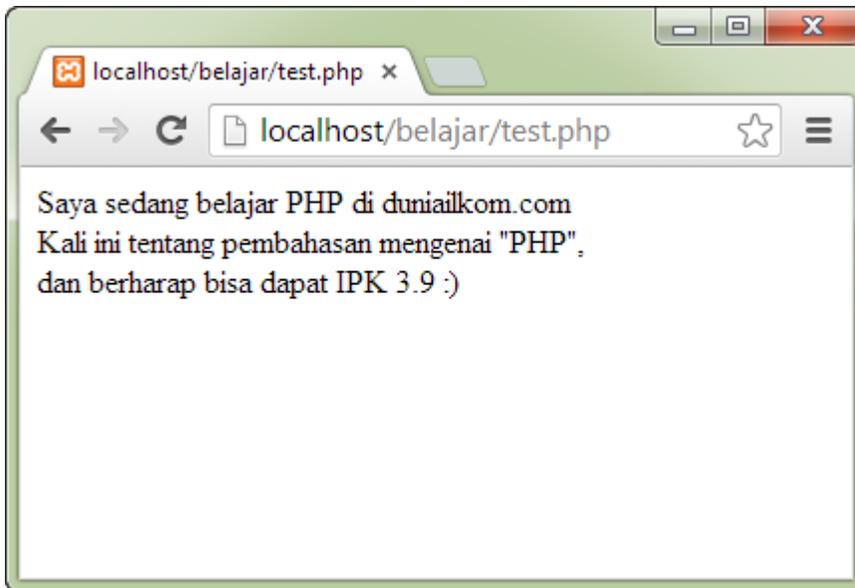
2.3.6.4 PENULISAN TIPE DATA STRING DENGAN HEREDOC

Cara penulisan tipe data string yang ketiga yaitu dengan fitur PHP yang disebut heredoc. Fitur ini digunakan untuk membuat tipe data string yang dapat berisi beberapa baris kalimat. Dibandingkan

dengan menggunakan single quote dan double quote, pembuatan string dengan heredoc tidak terlalu sering digunakan.

Agar lebih jelas, berikut adalah contoh penulisan tipe data string dengan heredoc:

```
<?php
$IPK=3.9;
$string1 = <<<end
Saya sedang belajar PHP
di duniaikom.com <br />
Kali ini tentang pembahasan
mengenai "PHP", <br /> dan berharap
bisa dapat IPK $IPK :)
end;
echo $string1;
?>
```



Mari kita bahas tentang cara penulisan Heredoc. Seperti yang terlihat dari contoh diatas, fitur Heredoc ditandai dengan tanda "<<<" untuk memulai string, lalu diikuti dengan karakter penanda akhir string. Dari contoh tersebut kata "end" pada awal string adalah penanda akhir string. Anda bebas mengganti kata "end" dengan kata atau karakter lain, sepanjang kata tersebut bisa dijamin tidak akan muncul didalam string.

Setelah karakter penanda string, baris pertama setelahnya adalah awal dari string. String ini dapat mencakup beberapa baris, sampai ditemukan karakter penanda akhir string yang kita definisikan di awal (yaitu kata "end"). Setelah ditemukan karakter penanda akhir string, maka pendefinisian string berakhir.

Perhatikan juga bahwa di dalam kalimat diatas, saya menggunakan karakter \n dan variabel \$IPK. Seluruh karakter ini diproses oleh PHP, sehingga mirip dengan fitur double quoted string.

Penting untuk diperhatikan bahwa setelah tanda penutup heredoc (di dalam contoh diatas adalah kata 'end') dan tanda titik koma ";", tidak boleh ada spasi atau karakter apapun. Jika anda menuliskan seperti berikut ini:

```
end ;
```

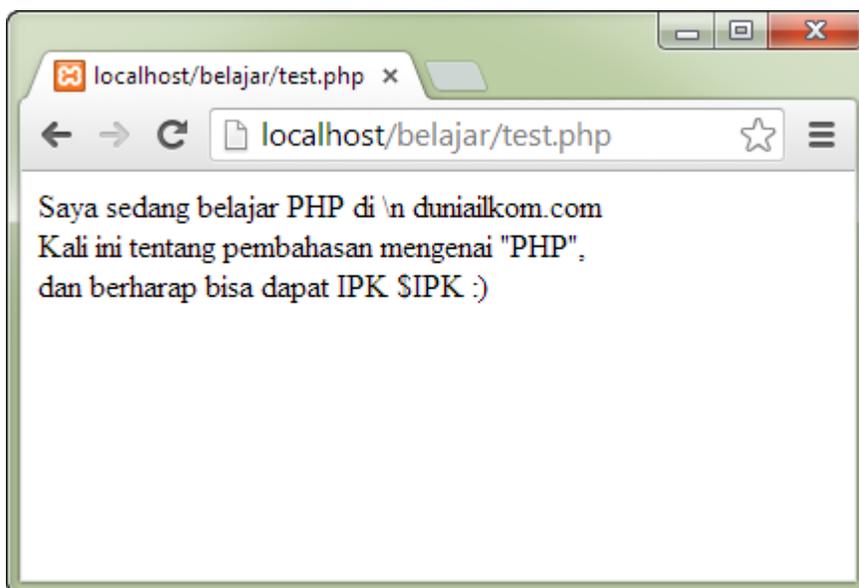
PHP akan mengeluarkan error: Parse error: syntax error, unexpected end of file.

2.3.6.5 PENULISAN TIPE DATA STRING DENGAN NOWDOC

Cara penulisan tipe data keempat dalam PHP yaitu dengan fitur Nowdoc. Fitur ini hampir sama dengan Heredoc, namun dengan pengecualian: karakter khusus dan variabel tidak akan diproses oleh PHP, atau mirip dengan single quoted string.

Berikut adalah contoh penulisan tipe data string menggunakan metode Nowdoc:

```
<?php
$IPK=3.9;
$string1 = <<< 'selesai'
Saya sedang belajar PHP
di \n duniaikom.com <br />
Kali ini tentang pembahasan
mengenai "PHP", <br /> dan berharap
bisa dapat IPK $IPK :)
selesai;
echo $string1;
?>
```



Jika dilihat sekilas, tidak ada perbedaan cara penulisan metode nowdoc dengan heredoc, namun perhatikan karakter penanda akhir string. Kali ini saya menggunakan karakter 'selesai' sebagai penanda akhir string. Dan yang membedakannya dengan heredoc adalah, nowdoc menambahkan single quoted untuk karakter penanda akhir string. Saya menulis 'selesai' (dengan tanda kutip satu) untuk mengawali string.

Dari tampilan yang dihasilkan, nowdoc memproses string sama dengan single quoted string, dimana karakter khusus dan variabel tidak diproses sama sekali, sehingga dalam tampilan akhir anda dapat melihat tanda \n dan variabel \$IPK ditulis sebagai string.

Dalam tutorial belajar PHP kali ini kita telah membahas 4 cara penulisan dan pendefinisian tipe data string, yaitu dengan single quoted, double quoted, heredoc, dan nowdoc. Metode penulisan string dengan heredoc dan nowdoc mungkin tidak akan sering anda jumpai.

2.3.6.6 PENGERTIAN DAN JENIS OPERATOR STRING DALAM PHP

Dalam PHP, hanya terdapat 1 jenis operator String, yakni operasi penyambungan (concatenation) string. Operator ini menggunakan karakter titik (.). Operator penyambungan string ini membutuhkan 2 inputan yang bertipe data string. Hasil dari operator ini adalah sebuah string yang terdiri dari sambungan kedua string tersebut.

Cara Penggunaan Operator String di dalam PHP

Berikut adalah contoh kode program cara penggunaan operator string dalam PHP:

```
<?php
$a = "Hello ";
$hasil = $a . "World!";
echo $hasil; // Hello World!
echo "<br />";
$a = "belajar ";
$b = "PHP ";
$c = "di Duniailkom.com";
$hasil= "Saya sedang ".$a.$b.$c;
echo $hasil; // Saya sedang belajar PHP di Duniailkom.com
?>
```

Pada kode program diatas, saya menyambung beberapa string sederhana menggunakan operator concatenation (tanda .).

Cara Alternatif: Penyambung string dengan kurung kurawal { }

Didalam PHP, tanda kurung kurawal (karakter { dan }) untuk variabel bisa berfungsi sebagai penyambung string. Contoh kode program diatas dapat juga ditulis menjadi:

```
<?php
$a = "Hello ";
$hasil = "{$a} World!";
echo $hasil; // Hello World!
echo "<br />";
$a = "belajar ";
$b = "PHP ";
$c = "di Duniailkom.com";
$hasil= "Saya sedang {$a}{$b}{$c}";
echo $hasil; // Saya sedang belajar PHP di Duniailkom.com
?>
```

Contoh diatas “memanfaatkan” sifat pendefinisian string menggunakan tanda kutip dua (double quote). Seperti yang telah kita bahas pada tutorial Mengenal Tipe Data String dan Cara Penulisan String dalam PHP, jika pendefinisian string menggunakan double quote (karakter “), maka setiap variabel akan diproses oleh PHP.

Namun kita tidak bisa menulis :

```
$hasil= "Saya sedang $a$b$c"; //akan menghasilkan error
```

Karena yang akan diproses PHP adalah 1 variabel saja, yakni \$a\$b\$c. sehingga kita perlu menambahkan tanda kurung kurawal (karakter { dan }) untuk memisahkan ketiga string menjadi {\$a}{\$b}{\$c}.

Beberapa situs referensi PHP, menyebutkan bahwa cara ini “lebih cepat” untuk menyambung string daripada menggunakan operator titik (.), Namun anda tidak akan melihat perbedaanya untuk kode program sederhana.

2.3.7 BOOLEAN

Tipe data boolean adalah tipe data paling sederhana dalam PHP dan juga dalam bahasa pemrograman lainnya. Tipe data ini hanya memiliki 2 nilai, yaitu true (benar) dan false (salah).

Tipe data boolean biasanya digunakan dalam operasi logika seperti kondisi if, dan perulangan (looping). Untuk penggunaan tipe data boolean akan kita pelajari pada waktu membahas tentang struktur pemrograman PHP.

2.3.7.1 CARA PENULISAN BOOLEAN DALAM PHP

Penulisan boolean cukup sederhana, karena hanya memiliki 2 nilai, yakni true atau false. Penulisan true atau false ini bersifat non-case sensitif, sehingga bisa ditulis sebagai true, True atau TRUE.

Berikut adalah contoh penulisan tipe data boolean:

```
<?php
$benar=true;
$salah=false;
echo "benar = $benar, salah = $salah";
// hasil output: benar = 1, salah =
?>
```

Jika anda menjalankan contoh kode PHP diatas, variabel \$benar akan ditampilkan dengan angka 1, sedangkan variabel \$false ditampilkan dengan string kosong (tanpa output). Hal ini karena jika ditampilkan menggunakan echo, tipe data boolean “dipaksa” berganti dengan tipe data string. (Lebih lanjut tentang konversi antar tipe data akan kita dalam Tutorial PHP: Cara Mengubah Tipe Data PHP.

2.3.7.2 KONVERSI TIPE DATA LAIN MENJADI BOOLEAN

Karena PHP adalah loosely typed language, atau bahasa pemograman yang tidak bertipe, sebuah variabel dapat di konversi menjadi tipe data lainnya.

Berikut adalah aturan tipe data boolean jika dikonversi dari tipe data lainnya:

- a) Integer 0, dianggap sebagai false.
- b) Float 0.0, dianggap sebagai false.
- c) String kosong ("") dan string "0" dianggap sebagai false.
- d) Array tanpa elemen, dianggap sebagai false.
- e) Objek dengan tanpa nilai dan fungsi, dianggap sebagai false.
- f) Nilai null, dianggap sebagai false.

Selain 6 kondisi diatas, sebuah variabel akan dikonversi menjadi true.

Berikut adalah contoh variabel dan nilai konversinya dalam boolean:

```
<?php
$x = FALSE; // false
$x = ""; // false
$x = " "; // true
$x = 1; // true
$x = -2; // true
$x = "belajar"; // true
$x = 3.14; // true
$x = array(); // false
$x = array(12); // true
$x = "false"; // true
```

```
?>
```

Perhatikan beberapa konversi diatas, string "" (string kosong) dianggap sebagai false, namun string " " (string dengan karakter spasi) adalah true. Juga string "0" dianggap false, namun string "false" dianggap true.

Kesalahan dalam kode program sering terjadi karena "konversi" dari tipe data lain menjadi boolean, sehingga sedapat mungkin kita membuat variabel boolean dengan nilai yang pasti dan tidak bergantung kepada aturan "konversi" boolean dari PHP.

Dalam tutorial kali ini, kita telah membahas tentang tipe data boolean PHP serta hasil boolean dari konversi tipe data lain. Tipe data boolean ini akan banyak digunakan dalam operasi logika seperti if.

2.3.8 ARRAY

Array (atau larik dalam bahasa Indonesia) bukanlah tipe data dasar seperti integer atau boolean, Array adalah sebuah tipe data bentukan yang terdiri dari kumpulan tipe data lainnya. Menggunakan array akan memudahkan dalam membuat kelompok data, serta menghemat penulisan dan penggunaan variabel. Misalkan kita butuh untuk menyimpan 10 nama mahasiswa, maka kode PHPnya jika tanpa menggunakan array adalah sebagai berikut:

```
<?php
$nama0="Andri";
$nama1="Joko";
$nama2="Sukma";
$nama3="Rina";
$nama4="Sari";
//... dst sampai $nama10
?>
```

Kode PHP seperti diatas tidak salah, tetapi kurang efektif karena kita membuat 10 variabel untuk 10 nama. Bagaimana jika kita butuh 100 nama? maka akan dibutuhkan 100 variabel \$nama. Pembuatan kode program diatas akan lebih rapi jika ditulis kedalam bentuk array, karena kita hanya membutuhkan 1 buah variabel saja untuk menampung banyak nilai. Berikut adalah contoh penggunaan array:

```
<?php
$nama = array (
    0=>"Andri",
    1=>"Joko",
    2=>"Sukma",
    3=>"Rina",
    4=>"Sari",)
//... dst sampai 10
?>
```

2.3.9 CARA PENULISAN ARRAY DALAM PHP

PHP mendukung beberapa cara penulisan array, salah satunya dengan menggunakan konstruktor array PHP (array language construct) sebagai berikut:

```
$nama_variabel = array(  
key => value,  
key2 => value2,  
key3 => value3,  
...  
)
```

Komponen array terdiri dari pasangan kunci (key) dan nilai (value). Key adalah penunjuk posisi dimana value disimpan. Perhatikan juga bahwa PHP menggunakan tanda panah (=>) untuk memberikan nilai kepada key. Dalam mengakses nilai dari array, kita menggunakan kombinasi \$nama_variabel dan nilai key-nya, dengan penulisan sebagai berikut:

```
$nama_variabel[key];
```

Berikut adalah contoh pengaksesan array dalam PHP:

```
<?php  
//pembuatan array  
$nama = array(  
    1=>"Andri",  
    2=>"Joko",  
    3=>"Sukma",  
    4=>"Rina",  
    5=>"Sari");  
//cara akses array  
echo $nama[1]; //Andri  
echo "<br />";  
echo $nama[2]; //Joko  
echo "<br />";  
echo $nama[3]; //Sukma  
?>
```

Dalam contoh diatas, saya menggunakan angka integer sebagai key (1,2,3...) dan string sebagai value (Andri, Joko, Sukma, ...). Selain mendefenisikan key secara langsung, PHP juga memperbolehkan penulisan array tanpa key, dan key itu secara otomatis akan diurutkan dari nilai 0, 1, 2, dst.

Berikut adalah contoh pendefenisian array tanpa key:

```
<?php  
// pembuatan array  
$nama = array("Andri", "Joko", "Sukma", "Rina", "Sari");  
// pengaksesan array  
echo $nama[1]; //Joko
```

```

echo "<br />";
echo $nama[2]; //Sukma
echo "<br />";
echo $nama[3]; //Rina
?>

```

Perhatikan bahwa sekarang, index atau key dari array dimulai dari angka 0, bukan 1. sehingga \$nama[1] berisi Joko. nama Andri berada di \$nama[0]. Dalam penggunaan array di dalam PHP, konsep “key” array dimulai dari angka 0 ini sangat penting untuk dipahami.

Selain menggunakan angka, key dalam PHP dapat berisi string atau boolean. Sedangkan untuk value dapat menyimpan berbagai tipe data seperti integer, float, string, boolean, bahkan array lainnya. Array seperti ini disebut juga dengan istilah “associate array”.

Berikut contoh penggunaan array dengan kombinasi tipe data.

```

<?php
// pembuatan array
$coba = array (
    2=>"Andri",
    "dua"=>"2",
    'tiga'=>3,
    true=>>true,
    9=>"sembilan",);
// pengaksesan array
echo $coba[2]; //Andri
echo "<br />";
echo $coba["dua"]; //2
echo "<br />";
echo $coba['tiga']; //3
echo "<br />";
echo $coba[true]; //1 (true di konversi menjadi 1)
echo "<br />";
echo $coba[9]; // sembilan
?>

```

Dari contoh diatas, saya membuat array \$coba dengan menggunakan berbagai tipe data untuk key dan value, yaitu dengan tipe data integer, string, dan boolean. Namun jika key di definisikan dengan tipe data boolean seperti pada baris ke-6, maka secara otomatis PHP akan mengkonversinya menjadi 1. Untuk PHP versi 5.4.x keatas, PHP menyediakan cara yang lebih singkat dalam pembuatan array, atau disebut dengan ‘short syntax array’. Berikut contoh penggunaannya:

```

<?php
// pembuatan array
$nama = ["Andri","Joko","Sukma","Rina","Sari"];
// pengaksesan array
echo $nama[1]; //Joko
echo "<br />";

```

```
echo $nama[2]; //Sukma
echo "<br />";
echo $nama[3]; //Rina
?>
```

Perhatikan bahwa kita tidak perlu membuat keyword 'array', tapi langsung membuat tanda kurung siku. Dan, seperti biasa, index key array dimulai dari 0. Sehingga pemanggilan \$nama[1] menghasilkan "Joko", bukan "Andri".

Dalam tutorial kali ini kita telah membahas cara pembuatan Mengenal Tipe Data Array dan Cara Penulisan Array dalam PHP. Array merupakan sebuah tipe data yang sangat berguna dalam pembuatan program nantinya (terutama untuk menampilkan hasil dari database) dan PHP menyediakan banyak fungsi untuk mendukung pemrosesan array, hal ini akan kita bahas dalam pembahasan tentang array dalam tutorial lanjutan.

2.4 OPERATOR PHP

Dalam bahasa pemrograman, terdapat istilah operand dan operator. Operand adalah nilai asal yang digunakan didalam proses operasi, sedangkan operator adalah instruksi yang diberikan untuk mendapatkan hasil dari proses tersebut.

Contohnya, operasi: 5+2. Angka 5 dan 2 adalah operand, sedangkan tanda tambah (karakter +) adalah operator. Beberapa operator bisa mengubah nilai dari operandnya sendiri, walaupun kebanyakan hanya sebagai penghubung antar operand. Operator di dalam PHP banyak meminjam contoh karakter dari bahasa C dan Perl.

2.4.1 JENIS OPERATOR BERDASARKAN JUMLAH OPERAND

Berdasarkan jumlah operand, operator dapat dibedakan menjadi 3, yaitu Operator Unary, Binary dan Ternary. Operator unary adalah operator yang hanya memiliki 1 operand, contohnya karakter - (tanda minus). Tanda minus digunakan membuat sebuah angka menjadi negatif, contohnya: -5, atau karakter + untuk menegaskan nilai positif, contohnya: +5. Operator binary adalah operator yang memiliki 2 operand. Operator jenis ini adalah yang paling banyak digunakan, misalkan 5×2, atau 10/3. Operator Ternary adalah operator yang memiliki 3 operand. Didalam PHP hanya dikenal 1 operator ternary, yaitu operator kondisi (? :). Kita akan mempelajari operator ini dalam tutorial selanjutnya.

2.4.2 URUTAN PRIORITAS OPERATOR DALAM PHP

Sama seperti membuat persamaan dalam matematika, operator dalam PHP juga memiliki urutan pemrosesan tersendiri. Misalkan terdapat kode program sebagai berikut:

```
$hasil1 = $a + $b/$c - $d;  
$hasil2 = $a AND $b || $c AND $d;
```

Program tersebut akan dieksekusi oleh PHP dengan melihat urutan prioritasnya. Urutan prioritas ini menetapkan seberapa “dekat” operator dengan kedua operand-nya. Sebagai contoh, $2+3*5$ hasilnya adalah 17, bukan 25. Hal ini karena operator perkalian (*) memiliki prioritas lebih tinggi daripada operator penambahan (+). Namun urutan prioritas ini dapat “dipaksa” dengan menggunakan tanda kurung, jika anda ingin menjumlahkan 2 dan 3 terlebih dahulu, maka operasi sebelumnya bisa ditulis menjadi $(2+3)*5$ yang hasilnya adalah 25. Berikut adalah tabel urutan prioritas operator dalam PHP. Operator paling atas lebih diprioritaskan dari pada operator dibawahnya, dan operator yang berada dalam baris yang sama memiliki urutan prioritas yang sama.

Urutan Proses	Operator	Keterangan
tidak-diurutkan	clone new	Pembuatan object
kiri	[] array()	Pembuatan array
kanan	++ -- ~ (int) (float) (string) (array) (object) (bool) @	Casting tipe data dan increment/decrement
tidak-diurutkan	instance of	Testing tipe data
kanan	!	Logika NOT
kiri	* / %	Operator matematika
kiri	+ - .	Operator matematika dan string
kiri	<< >>	Operator bitwise
tidak-diurutkan	< <= > >=	Operator perbandingan
tidak-diurutkan	== != === !== <>	Operator perbandingan
kiri	&	Operator bitwise dan referensi
kiri	^	Operator bitwise
kiri		Operator bitwise
kiri	&&	Operator logika
kiri		Operator logika
kiri	? :	Operator kondisi
kanan	= += -= *= /= .= %= &= = ^= <<= >>= =>	Operator assignment
kiri	and	Operator logika
kiri	xor	Operator logika
kiri	or	Operator logika
kiri	,	Berbagai fungsi Duniailkom.com

Dari tabel diatas, terdapat beberapa operator yang berada dalam 1 baris. Bagaimana PHP memproses operator-operator yang memiliki urutan prioritas sama? Di dalam PHP, operator tersebut akan diproses berdasarkan kolom arah proses. Kolom arah proses (atau dalam manual PHP disebut dengan Associativity) digunakan untuk melihat bagaimana arah proses operator dijalankan. Misalkan operator kurang (-), di dalam tabel dapat dilihat bahwa operator kurang (-) memiliki arah proses “kiri”, sehingga operasi $5 - 3 - 1$ oleh PHP diproses dari kiri ke kanan. $5 - 3 - 1$ diproses menjadi $(5 - 3) - 1$, dan hasilnya adalah 1. Namun di dalam tabel, operator “=” memiliki arah proses “kanan”, sehingga $\$a = \$b = \$c$, akan diproses dari kanan terlebih dahulu, menjadi $\$a = (\$b = \$c)$. Jika arah proses tersebut “non-arah”, berarti operator itu tidak bisa digunakan secara berdampingan. Misalkan $4 < 6 > 2$, tidak dapat diproses oleh PHP, namun $1 <= 1 == 1$ bisa diproses karena operator == memiliki urutan prioritas lebih rendah daripada <=. Selain untuk memaksakan urutan prioritas, penggunaan tanda kurung juga akan memudahkan pembacaan program, bahkan ketika tidak diperlukan. Misalkan $\$a \text{ AND } \$b \text{ OR } \$c$, akan lebih mudah

dimengerti ketika ditulis menjadi ($\$a$ AND $\$b$) OR $\$c$, walaupun sebenarnya operator AND memiliki urutan prioritas lebih tinggi daripada OR.

Karena sifat variabel dalam PHP yang tidak bertipe (Loosely Typed Language), dalam pembuatan program PHP sebuah tipe variabel dapat “berubah” menjadi tipe lainnya. Perubahan ini bergantung operator yang digunakan. Seperti yang kita lihat pada saat pembahasan tentang tipe data boolean, tipe data string “aku”, dapat menjadi tipe data boolean TRUE.

Perhatikan contoh kode berikut:

```
<?php
$a= 5; $b=8; $c=4.5;
$hasil1=$a+$b;
$hasil2=$a+$c;
$hasil3=$a.$b;
echo "\$hasil1: $hasil1 <br/>"; //13
echo "\$hasil2: $hasil2 <br/>"; //9.5
echo "\$hasil3: $hasil3 <br/>"; //"58"
?>
```

Dari contoh kode diatas, variabel $\$a$ dan $\$b$ bertipe integer, dan variabel $\$c$ bertipe float. Namun variabel $\$hasil1$, $\$hasil2$ dan $\$hasil3$ akan bertipe integer, float, dan string secara berurutan.

Salah satu aturan di dalam PHP, jika operator penyambungan string (karakter titik) digunakan untuk tipe data integer, secara otomatis PHP akan mengkonversinya menjadi string, sehingga:

```
5(integer) + 8(integer) = 13(integer)
5(integer) . 8(integer) = 58(string)
```

Proses konversi ini dilakukan secara otomatis oleh PHP, sehingga kita memerlukan sebuah fungsi untuk mengetahui secara lebih detail tipe data dan nilai sebuah variabel. Untuk keperluan inilah PHP menyediakan fungsi `var_dump()`.

2.4.3 VAR_DUMP() DAN PRINT_R()

Karena sifat variabel dalam PHP yang tidak bertipe (Loosely Typed Language), dalam pembuatan program PHP sebuah tipe variabel dapat “berubah” menjadi tipe lainnya. Perubahan ini bergantung operator yang digunakan. Seperti yang kita lihat pada saat pembahasan tentang tipe data boolean, tipe data string “aku”, dapat menjadi tipe data boolean TRUE.

Perhatikan contoh kode berikut:

```
<?php
$a= 5; $b=8; $c=4.5;
```

```

$hasil1=$a+$b;
$hasil2=$a+$c;
$hasil3=$a.$b;
echo "\$hasil1: $hasil1 <br/>"; //13
echo "\$hasil2: $hasil2 <br/>"; //9.5
echo "\$hasil3: $hasil3 <br/>"; //"58"
?>

```

Dari contoh kode diatas, variabel \$a dan \$b bertipe integer, dan variabel \$c bertipe float. Namun variabel \$hasil1, \$hasil2 dan \$hasil3 akan bertipe integer, float, dan string secara berurutan. Salah satu aturan di dalam PHP, jika operator penyambungan string (karakter titik) digunakan untuk tipe data integer, secara otomatis PHP akan mengkonversinya menjadi string, sehingga:

```

5(integer) + 8(integer) = 13(integer)
5(integer) . 8(integer) = 58(string)

```

Proses konversi ini dilakukan secara otomatis oleh PHP, sehingga kita memerlukan sebuah fungsi untuk mengetahui secara lebih detail tipe data dan nilai sebuah variabel. Untuk keperluan inilah PHP menyediakan fungsi `var_dump()`.

2.4.3.1 CARA PENULISAN FUNGSI VAR_DUMP()

Untuk memastikan tipe data dari sebuah variabel, PHP menyediakan fungsi yang sangat berguna, terutama untuk proses pengujian dan pencarian kesalahan (debugging), yakni fungsi `var_dump()`.

Fungsi `var_dump()` membutuhkan inputan variabel yang akan diperiksa. Berikut contoh penggunaan fungsi `var_dump()`:

```

<?php
$a= 5; $b=8; $c=4.5;
$hasil1=$a+$b;
$hasil2=$a+$c;
$hasil3=$a.$b;
echo "\$hasil1: "; var_dump($hasil1); //int(13)
echo "<br \>"; //
echo "\$hasil2: "; var_dump($hasil2); //float(9.5)
echo "<br \>";
echo "\$hasil3: "; var_dump($hasil3); //string(2) "58"
?>

```

Dari tampilan hasil kode PHP tersebut, fungsi `var_dump()` selain menampilkan hasil variabel, juga memperlihatkan jenis tipe dari variabel tersebut. Fitur ini akan sangat berguna dalam proses pembuatan kode program PHP yang lebih rumit untuk menghindari kesalahan.

Perhatikan pula hasil dari `var_dump($hasil3)`, hasil fungsi `var_dump` adalah `string(2) "58"`. Angka 2 disini menjelaskan panjang dari variabel string tersebut.

2.4.4 OPERATOR ARITMATIKA

Operator Aritmatika adalah operator matematis yang terdiri dari operator penambahan, pengurangan, perkalian, pembagian, modulus, plus, dan minus.

2.4.4.1 JENIS OPERATOR ARITMATIKA DALAM PHP

Didalam PHP terdapat 7 jenis operator aritmatika, berikut ke tujuh operator tersebut:

Contoh	Nama Operator	Hasil
<code>+\$a</code>	Positif	Nilai positif dari \$a Duniailkom.com
<code>-\$a</code>	Negatif	Nilai negatif dari \$a
<code>\$a + \$b</code>	Penambahan	Total dari \$a dan \$b
<code>\$a - \$b</code>	Pengurangan	Selisih dari \$a dan \$b
<code>\$a * \$b</code>	Perkalian	Hasil Kali dari \$a dan \$b
<code>\$a / \$b</code>	Div/Pembagian	Hasil Bagi dari \$a dan \$b
<code>\$a % \$b</code>	Mod/Sisa hasil bagi	Sisa dari pembagian \$a / \$b

Kebanyakan operator aritmatika dalam PHP bertipe binary yakni membutuhkan 2 operand, kecuali operator minus (-) dan plus (+) yang merupakan operator tipe unary (hanya membutuhkan 1 operand).

Dari ke 7 operator aritmatika dalam PHP tersebut, operator modulus (`$a % $b`) mungkin terdengar baru. Operator ini menghasilkan sisa hasil bagi dari hasil pembagian. Misalkan `10 % 3`, hasilnya adalah 1. Biasanya operator modulus ini digunakan bersama-sama dengan operator pembagian (`/`).

2.4.4.2 CARA PENGGUNAAN OPERATOR ARITMATIKA DI DALAM PHP

Penggunaan operator aritmatika di dalam PHP relatif mudah, karena kita telah terbiasa dengan operator ini. Berikut adalah contoh kode program, cara penggunaan operator aritmatika dalam PHP:

```
<?php
$hasil1= -3;
$hasil2=3+5;
$hasil3=8-4.5;
$hasil4=2*5;
$hasil5=3+8/5-3;
$hasil6=10 % 4;
```

```

echo "$hasil1:"; var_dump($hasil1); // $hasil1:int(-3)
echo "<br \>";
echo "$hasil2:"; var_dump($hasil2); // $hasil2:int(8)
echo "<br \>";
echo "$hasil3:"; var_dump($hasil3); // $hasil3:float(3.5)
echo "<br \>";
echo "$hasil4:"; var_dump($hasil4); // $hasil4:int(10)
echo "<br \>";
echo "$hasil5:"; var_dump($hasil5); // $hasil5:float(1.6)
echo "<br \>";
echo "$hasil6:"; var_dump($hasil6); // $hasil6:int(2)
?>

```

Pada kode program diatas, saya menggunakan fungsi `var_dump()` untuk menampilkan hasil perhitungan, sehingga kita bisa melihat tipe data dari masing-masing variabel. Dari hasil `var_dump()`, terlihat bahwa variabel `$hasil3` dan `$hasil5` bertipe float. Hal ini dikarenakan perhitungan aritmatika pada baris ke-4 dan ke-6 menghasilkan angka desimal, sehingga secara otomatis variabel tersebut tidak dapat ditampung sebagai integer, melainkan harus float.

Namun jika hasil operasi matematis tersebut menghasilkan bilangan bulat, PHP akan menyimpannya sebagai tipe data int (integer), seperti variabel `$hasil1`, `$hasil2`, `$hasil4` dan `$hasil6`. Pada perhitungan baris ke-6 yaitu persamaan `$hasil5=3+8/5-3`, hasilnya adalah 1.6. Hal ini karena operator pembagian memiliki prioritas lebih tinggi daripada operator tambah dan kurang. Operasi `3+8/5-3` dikerjakan oleh PHP sebagai `(3+(8/5))-3`. Namun untuk hal ini, disarankan menggunakan tanda kurung secara tertulis agar memudahkan dalam membaca alur program, dari pada bergantung kepada aturan prioritas operator PHP. Namun jika tidak ditegaskan dengan menggunakan tanda kurung, urutan prioritas operator matematis dalam PHP mengikuti aturan tabel yang kita bahas pada tutorial Pengertian Operand, Operator dan Urutan Operator dalam PHP

2.4.5 OPERATOR STRING

Dalam PHP, hanya terdapat 1 jenis operator String, yakni operasi penyambungan (concatenation) string. Operator ini menggunakan karakter titik (`.`). Operator penyambungan string ini membutuhkan 2 inputan yang bertipe data string. Hasil dari operator ini adalah sebuah string yang terdiri dari sambungan kedua string tersebut.

2.4.5.1 CARA PENGGUNAAN OPERATOR STRING DI DALAM PHP

Berikut adalah contoh kode program cara penggunaan operator string dalam PHP:

```

<?php
$a = "Hello ";

```

```

$hasil = $a . "World!";
echo $hasil; // Hello World!
echo "<br />";
$a = "belajar ";
$b = "PHP ";
$c = "di Duniailkom.com";
$hasil= "Saya sedang ".$a.$b.$c;
echo $hasil; // Saya sedang belajar PHP di Duniailkom.com
?>

```

Pada kode program diatas, saya menyambung beberapa string sederhana menggunakan operator concatenation (tanda .).

2.4.5.2 CARA ALTERNATIF: PENYAMBUNG STRING DENGAN KURUNG KURAWAL { }

Didalam PHP, tanda kurung kurawal (karakter { dan }) untuk variabel bisa berfungsi sebagai penyambung string Contoh kode program diatas dapat juga ditulis menjadi:

```

<?php
$a = "Hello ";
$hasil = "{$a} World!";
echo $hasil; // Hello World!
echo "<br />";
$a = "belajar ";
$b = "PHP ";
$c = "di Duniailkom.com";
$hasil= "Saya sedang {$a}{$b}{$c}";
echo $hasil; // Saya sedang belajar PHP di Duniailkom.com
?>

```

Contoh diatas “memanfaatkan” sifat pendefinisian string menggunakan tanda kutip dua (double quote). Seperti yang telah kita bahas pada tutorial Mengenal Tipe Data String dan Cara Penulisan String dalam PHP, jika pendefinisian string menggunakan double quote (karakter “), maka setiap variabel akan diproses oleh PHP. Namun kita tidak bisa menulis :

```

$hasil= "Saya sedang $a$b$c"; //akan menghasilkan error

```

Karena yang akan diproses PHP adalah 1 variabel saja, yakni \$a\$b\$c. sehingga kita perlu menambahkan tanda kurung kurawal (karakter { dan }) untuk memisahkan ketiga string menjadi {\$a}{\$b}{\$c}. Beberapa situs referensi PHP, menyebutkan bahwa cara ini “lebih cepat” untuk menyambung string daripada menggunakan operator titik (.), Namun anda tidak akan melihat perbedaanya untuk kode program sederhana.

2.4.6 OPERATOR LOGIKA

Operator Logika adalah operator yang digunakan untuk membandingkan 2 kondisi logika, yaitu logika benar (TRUE) dan logika salah (FALSE). Operator logika sering digunakan untuk kondisi IF, atau untuk keluar dari proses perulangan (looping).

Jenis operand dalam operator logika ini adalah variabel dengan tipe boolean. Namun jika operand bukan boolean, akan “dikonversi” menjadi boolean oleh PHP (aturan “konversi” ini telah kita bahas pada tutorial tentang tipe data boolean PHP).

2.4.6.1 JENIS-JENIS OPERATOR LOGIKA DALAM PHP

Jenis-jenis operator logika dalam PHP dapat dilihat dari tabel berikut:

Contoh	Nama Operator	Hasil
<code>\$a and \$b</code>	AND	TRUE jika \$a dan \$b sama-sama bernilai TRUE.
<code>\$a && \$b</code>	AND	TRUE jika \$a dan \$b sama-sama bernilai TRUE.
<code>\$a or \$b</code>	OR	TRUE jika salah satu dari \$a atau \$b adalah TRUE.
<code>\$a \$b</code>	OR	TRUE jika salah satu dari \$a atau \$b adalah TRUE.
<code>\$a xor \$b</code>	XOR	TRUE jika salah satu dari \$a atau \$b adalah TRUE, tetapi bukan keduanya.
<code>! \$a</code>	NOT	TRUE jika \$a=FALSE. Duniailkom.com

Perbedaan dari operator AND dengan `&&`, dan OR dengan `||` terkait dengan cara penulisan dan aturan “kekuatan” operator. Operator `&&` dan `||` memiliki “kekuatan” lebih tinggi dari pada AND dan OR, sehingga baris perintah: `$a AND $b || $c`, akan dieksekusi oleh PHP menjadi `$a AND ($b || $c)`.

Dari tabel diatas, saya hanya memberikan hasil untuk kondisi TRUE, maka selain kondisi tersebut, hasilnya adalah FALSE.

2.4.6.2 CARA PENGGUNAAN OPERATOR LOGIKA DI DALAM PHP

Berikut adalah contoh kode program, cara penggunaan operator logika dalam PHP:

```
<?php
$hasil1 = true and false;
echo '$hasil1 = ';
echo var_dump($hasil1)."<br/>"; // $hasil1 = bool(true)
$hasil2 = (true and false);
echo '$hasil2 = ';
echo var_dump($hasil2)."<br/>"; // $hasil2 = bool(false)
$hasil3 = (true xor false);
echo '$hasil3 = ';
echo var_dump($hasil3)."<br/>"; // $hasil3 = bool(true)
$hasil4 = (false or true && false);
echo '$hasil4 = ';
```

```
echo var_dump($hasil4)."<br/>"; // $hasil4 = bool(false)
$a=true;
$b=false;
$hasil5 = ($a and $b || $a or b);
echo '$hasil5 = ';
echo var_dump($hasil5); // $hasil5 = bool(true)
?>
```

Saya akan membahas kode program diatas:

Pada baris 2, operasi logika yang dijalankan adalah \$hasil1 = true and false, yang harusnya \$hasil1 akan bernilai false (berdasarkan prinsip operator and: jika salah satu saja ada yang false, maka hasilnya adalah false) Namun seperti yang terlihat dalam tampilan saat program dijalankan, variabel \$hasil1 bernilai true!. Apa yang sebenarnya terjadi? Hal ini kembali kepada prinsip urutan prioritas operator.

Jika anda perhatikan tabel urutan operator pada tutorial urutan operator dalam PHP, operator assignment (pendefenisian variabel) yaitu menggunakan tanda sama dengan (=) memiliki priotitas lebih tinggi dari pada operator logika and. Sehingga yang sebenarnya diproses adalah (\$hasil1 = true) and false, sehingga \$hasil1 akan bernilai true. Kesalahan pemograman seperti ini akan sulit dideteksi, sehingga anda disarankan menggunakan tanda kurung untuk menegaskan urutan program.

Pada baris ke-6, untuk variabel \$hasil2 saya mengulangi operasi yang sama dengan baris 2, namun kali ini dengan menggunakan tanda kurung untuk memberitahukan kepada PHP bahwa operasi logikalalah yang harus dijalankan pertama kali, baru setelah itu proses assigment yang kali ini berjalan sebagaimana harusnya (true and false menghasilkan false). Dan variabel \$hasil2 bernilai false.

Pada baris ke-10, variabel \$hasil3 bernilai true, karena operator xor akan menghasilkan true apabila salah satu operand bernilai true.

Untuk baris ke-14, \$hasil4 = (false or true && false), urutan proses operasi dimulai terlebih dahulu pada operator && karena memiliki urutan prioritas lebih tinggi daripada operator or, sehingga yag diproses oleh PHP adalah (false or (true && false)), dan menghasilkan nilai false.

Pada contoh terakhir baris ke-20, operator || akan diproses terlebih dahulu, sehingga persamaanya menjadi \$hasil5 = (\$a and (\$b || \$a) or b).

Dalam membuat operasi logika ini, sedapat mungkin untuk mengujinya terlebih dahulu, karena kesalahan program pada logika akan sulit terdeteksi.

2.4.6.3 PENGERTIAN PRINSIP SHORT CIRCUIT DALAM OPERASI LOGIKA PHP

PHP menjalankan operasi logika dengan prinsip **short-circuit**, yaitu jika dengan memeriksa satu perintah saja sudah didapati hasil logikanya, maka perintah lain tidak akan dijalankan. Contohnya:

```
$hasil = $a AND $b AND $c AND $d;
```

Jika pada saat program dijalankan **\$a** sudah bernilai **FALSE**, maka variabel **\$b**, **\$c** dan **\$d** tidak akan diperiksa lagi, karena apapun nilai variabel tersebut, hasilnya akan tetap FALSE. Hal ini akan berguna untuk kasus-kasus tertentu, seperti contoh berikut:

```
<?php
$result = fopen($filename) or exit();
?>
```

Fungsi **exit()** dalam kode PHP tersebut (yang akan membuat program PHP berhenti diproses) tidak akan dijalankan selama **fopen(\$filename)** bernilai true. Fungsi short-circuit ini sering digunakan dalam contoh-contoh aplikasi PHP.

2.4.7 OPERATOR PERBANDINGAN

Sesuai dengan namanya, operator perbandingan membandingkan nilai dari 2 operand. Hasilnya selalu salah satu dari TRUE atau FALSE. Hasil perbandingan akan bernilai TRUE jika kondisi perbandingan tersebut benar, atau FALSE jika kondisinya salah.

Operand untuk operator perbandingan ini bisa berupa tipe data angka (integer atau float), maupun bertipe string. Operator perbandingan akan memeriksa nilai dan (untuk beberapa operator) juga tipe data dari operand.

2.4.7.1 JENIS-JENIS OPERATOR PERBANDINGAN DALAM PHP

Jenis-jenis dari operator perbandingan dalam PHP dapat dilihat dari tabel dibawah ini:

Contoh	Nama Operator	Hasil
<code>\$a == \$b</code>	Sama dengan	TRUE jika \$a sama dengan \$b (tanpa melihat tipe data)
<code>\$a === \$b</code>	Identik dengan	TRUE jika \$a sama dengan \$b, dan memiliki tipe data yang sama
<code>\$a != \$b</code>	Tidak sama dengan	TRUE jika \$a tidak sama dengan \$b (tanpa melihat tipe data)
<code>\$a <> \$b</code>	Tidak sama dengan	TRUE jika \$a tidak sama dengan \$b (tanpa melihat tipe data)
<code>\$a !== \$b</code>	Tidak identik dengan	TRUE jika \$a tidak sama dengan \$b, dan memiliki tipe data yang tidak sama
<code>\$a < \$b</code>	Kurang dari	TRUE jika \$a kurang dari \$b
<code>\$a > \$b</code>	Lebih dari	TRUE jika \$a lebih dari \$b
<code>\$a <= \$b</code>	Kurang dari atau sama dengan	TRUE jika \$a kurang dari atau sama dengan \$b
<code>\$a >= \$b</code>	Lebih dari atau sama dengan	TRUE jika \$a lebih dari atau sama dengan \$b Duniailkom.com

Dikarenakan operasi perbandingan dapat memiliki operan berupa angka dan string, PHP memiliki aturan sebagai berikut:

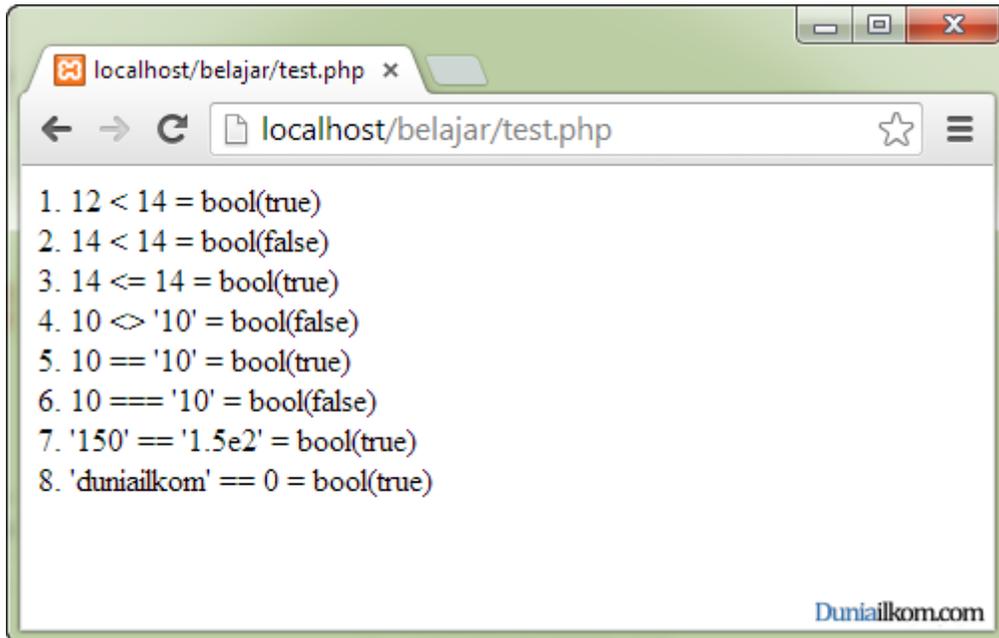
Tipe Operan Pertama	Tipe Operan Kedua	Proses Perbandingan
Angka	Angka	Angka
String berupa angka	String berupa angka	Angka
String berupa angka	Number	Angka
String berupa angka	String yang bukan angka	Angka
String yang bukan angka	Angka	Urutan Huruf
String yang bukan angka	String yang bukan angka	Urutan Huruf
Boolean	Boolean	FALSE lebih kecil dari TRUE
Boolean	Tipe apa saja	Operand kedua dikonversi menjadi Boolean
Array	Array	Array dengan data yang paling sedikit dianggap lebih kecil
Array	Tipe apa saja	Array selalu lebih besar
Objek	Tipe apa saja	Objek selalu lebih besar Duniailkom.com

Dari tabel diatas dapat dilihat bahwa selain membandingkan **angka** dengan **angka**, PHP juga membolehkan perbandingan **angka** dengan **string**, **array**, bahkan **objek**. Namun dalam prakteknya kita akan sering membandingkan angka.

2.4.7.2 CARA PENGGUNAAN OPERATOR PERBANDINGAN DI DALAM PHP

Berikut adalah beberapa contoh penggunaan **operator perbandingan** dalam PHP:

```
<?php
echo "1. 12 < 14 = "; var_dump(12<14); // bool(true)
echo "<br />";
echo "2. 14 < 14 = "; var_dump(14<14); // bool(false)
echo "<br />";
echo "3. 14 <= 14 = "; var_dump(14<=14); // bool(true)
echo "<br />";
echo "4. 10 <> '10' = "; var_dump(10<>'10'); // bool(false)
echo "<br />";
echo "5. 10 == '10' = "; var_dump(10=='10'); // bool(true)
echo "<br />";
echo "6. 10 === '10' = "; var_dump(10==='10'); // bool(false)
echo "<br />";
echo "7. '150' == '1.5e2' = "; var_dump('150'=='1.5e2'); // bool(true)
echo "<br />";
echo "8. 'duniailkom' == 0 = "; var_dump('duniailkom'==0); // bool(true)
echo "<br />";
?>
```

A screenshot of a web browser window displaying the output of a PHP script. The browser's address bar shows 'localhost/belajar/test.php'. The main content area lists eight comparison operations and their boolean results:

```
1. 12 < 14 = bool(true)
2. 14 < 14 = bool(false)
3. 14 <= 14 = bool(true)
4. 10 <> '10' = bool(false)
5. 10 == '10' = bool(true)
6. 10 === '10' = bool(false)
7. '150' == '1.5e2' = bool(true)
8. 'duniaikom' == 0 = bool(true)
```

The browser's footer shows 'Duniaikom.com'.

Dalam contoh kode PHP diatas, saya menggunakan fungsi `var_dump()` untuk melihat hasil dari perbandingan. Contoh 1 sampai 3 merupakan operasi perbandingan aritmatika biasa.

Pada contoh ke 4, `10 <> '10' = FALSE`, karena PHP menganggap kedua operand adalah sama, sehingga jika diberikan operator tidak sama dengan (`<>`), maka hasilnya **FALSE**. Perhatikan bahwa tipe data kedua angka berbeda, saya menambahkan tanda petik untuk membuah **string** '10', namun PHP mengkonversinya menjadi **integer** 10 ketika membandingkan (sesuai aturan tabel diatas).

Pada contoh ke 5, `10 == '10' = TRUE`, karena alasan yang sama dengan penjelasan contoh ke 4. String '10' dikonversi terlebih dahulu menjadi angka, lalu dibandingkan, sehingga hasilnya menjadi **TRUE**.

Untuk contoh ke 6, `10 === '10' = FALSE`, karena operator `===` selain membandingkan nilai, juga membandingkan tipe data dari opearand, sehingga **string** '10' dianggap tidak sama denganinteger 10. Hal ini berbeda dengan contoh ke 5.

Untuk contoh ke 7, `'150' == '1.5e2' = TRUE`, karena seluruh string yang berupa angka dikonversi menjadi angka terlebih dahulu, dan `'1.5e2'` adalah penulisan scientific dari `1,5 x 10^2`, yang hasilnya adalah 150. Namun sama seperti contoh ke 6, jika kita mengganti operator `'=='` dengan `'==='`, maka hasilnya akan **FALSE**.

Pada contoh terakhir, no 8. Saya membandingkan string dengan angka, sehingga string `'duniaikom'` dikonversi menjadi integer bernilai **0**, dan baru disamakan dengan operan kedua, yakni angka 0. Karena `0==0`, maka hasilnya = **TRUE**.

2.4.8 OPERATOR INCREMENT

Operator Increment dan Decrement adalah penyebutan untuk operasi seperti \$a++, dan \$a-. Jika anda telah mempelajari bahasa pemrograman lain, operasi increment dan decrement ini sering digunakan dalam perulangan (looping).

Increment digunakan untuk menambah variabel sebanyak 1 angka, sedangkan decrement digunakan untuk mengurangi variabel sebanyak 1 angka. Penulisannya menggunakan tanda tambah 2 kali untuk increment, dan tanda kurang 2 kali untuk decrement. Penempatan tanda tambah atau kurang ini boleh diawal, atau diakhir variabel, namun keduanya memiliki perbedaan, sehingga terdapat 4 jenis increment dan decrement dalam PHP.

2.4.8.1 JENIS OPERATOR INCREMENT DAN DECREMENT DALAM PHP

Berikut adalah tabel 4 jenis operator **increment** dan **decrement** dalam PHP:

Contoh	Nama	Hasil
++\$a	Pre-increment	Tambah nilai \$a sebanyak 1, lalu kirim nilai \$a
\$a++	Post-increment	Kirim nilai \$a, lalu tambah nilai \$a sebanyak 1
--\$a	Pre-decrement	Kurangi nilai \$a sebanyak 1, lalu kirim nilai \$a
\$a--	Post-decrement	Kirim nilai \$a, lalu kurangi nilai \$a sebanyak 1

Dari tabel diatas terlihat bahwa terdapat 2 jenis **increment**, yaitu **Pre-increment**, dan **Post-Increment**, dan 2 jenis **decrement**, yaitu **Pre-decrement** dan **Post-decrement**. Perbedaan keduanya terletak pada posisi mana tanda tambah atau kurang diletakkan.

2.4.8.2 CARA PENGGUNAAN OPERATOR INCREMENT DAN DECREMENT

Untuk memahami cara penggunaan operator **increment** dan **decrement**, berikut contoh kode program PHP:

```
<?php
echo "<h3>Postincrement</h3>";
$a = 5;
echo "$a = $a <br />";
echo "$a akan bernilai 5: " . $a++ . " ($a++)<br />";
echo "$a akan bernilai 6: " . $a . "<br />";

echo "<h3>Preincrement</h3>";
$a = 5;
echo "$a = $a <br />";
echo "$a akan bernilai 6: " . ++$a . " (++$a)<br />";
echo "$a akan bernilai 6: " . $a . "<br />";

echo "<h3>Postdecrement</h3>";
```

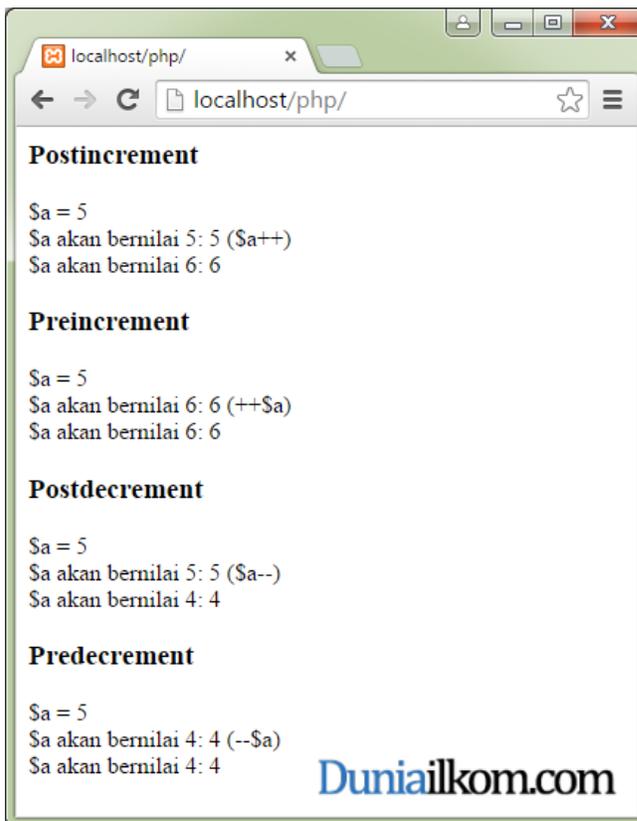
```

$a = 5;
echo "$a = $a <br />";
echo "$a akan bernilai 5: " . $a-- . " ($a--)<br />";
echo "$a akan bernilai 4: " . $a . "<br />";

echo "<h3>Predecrement</h3>";
$a = 5;
echo "$a = $a <br />";
echo "$a akan bernilai 4: " . --$a . " (--$a)<br />";
echo "$a akan bernilai 4: " . $a . "<br />";
?>

```

Contoh kode program diatas terlihat agak rumit, namun sebagian besar hanyalah pengaturan format agar hasilnya tampil rapi seperti berikut ini:



Terlihat bahwa **Post-increment (\$a++)**, akan memberikan hasilnya dulu, baru menambahkan nilai variabel **\$a** sebanyak 1 angka, namun dengan **Pre-increment**, **\$a** akan ditambahkan 1 angka, baru nilainya ditampilkan. Begitu juga halnya dengan operasi **Post-decrement** dan **Pre-decrement**.

2.4.9 OPERATOR ASSIGNMENT

Operator assignment adalah operator untuk *menambahkan*, atau *memasukkan* sebuah nilai kedalam **variabel**. PHP memiliki 3 jenis **operator assignment**, dan kita telah menggunakan 2 diantaranya, yaitu tanda = untuk mengdefinisikan **variabel**, dan

tanda => untuk mengisi nilai dari **array**.

2.4.9.1 JENIS-JENIS OPERATOR ASSIGNMENT DALAM PHP

PHP mengenal 3 jenis **operator assignment**, yaitu *Assignment dengan Nilai (Assignment by Value)*, *Assignment Array*, dan *Assignment dengan Referensi (Assignment by Reference)*.

2.4.9.2 PENGERTIAN OPERATOR ASSIGNMENT DENGAN NILAI (ASSIGNMENT BY VALUE)

Assignment dengan Nilai atau dalam istilah pemrograman dikenal dengan **Assignment by Value**, adalah proses pemberian nilai kedalam sebuah variabel dengan meng-copy nilai atau value dari variabel

lain. PHP menggunakan tanda sama dengan (=) untuk **Assignment by Value**. Secara tidak sadar, kita telah banyak menggunakan operator ini dalam berbagai contoh pada tutorial sebelumnya. Pada saat menggunakan operator "=", PHP meng-copy nilai (atau **value**), dari sisi kanan operator ke sisi kiri.

Operator **Assignment by Value** (dan juga operator assignment lainnya) memiliki **arah proses kanan** (dapat dilihat dalam tabel urutan proses pada [Tutorial PHP: Pengertian Operand, Operator dan Urutan Operator dalam PHP](#)) sehingga proses pemberian nilai kepada sebuah variabel dimulai dari **kanan**, seperti contoh berikut:

```
<?php
$a = 20;
$b = 15;
$c = 5;
echo "\$a = $a, \$b = $b, \$c = $c";
echo "<br />";
// hasil proses: $a = 20, $b = 15, $c = 5
$a = $b = $c+5;
echo "\$a = $a, \$b = $b, \$c = $c";
// hasil proses: $a = 10, $b = 10, $c = 5
?>
```

Pada contoh kode PHP diatas, saya mendefenisikan 3 variabel: **\$a**, **\$b**, dan **\$c** dengan operator **assignment by value**. Perhatikan pada baris terakhir, dimana saya menuliskan kode **\$a = \$b = \$c+5**, yang urutan pemrosesannya di mulai dari **kanan** ke **kiri**, sehingga yang diproses oleh PHP menjadi: **\$a = (\$b = (\$c+5))**.

2.4.9.3 PENGERTIAN OPERATOR ASSIGNMENT ARRAY

Operator **Assignment array** adalah operator assignment untuk menginput nilai kedalam array. Operator ini menggunakan tanda panah (=>). Kita telah membahas cara pembuatan, dan penggunaan operator ini pada tutorial Mengenal Tipe Data Array dan Cara Penulisan Array dalam PHP.

2.4.9.4 PENGERTIAN ASSIGNMENT DENGAN REFERENSI (ASSIGNMENT BY REFERENCE)

Assignment dengan referensi atau dalam istilah programmingnya: **Assignment by Reference**, adalah operator **assignment** khusus yang digunakan untuk **men-copy nilai referensi** dari sebuah *variabel*.

Pengalaman saya, assignment dengan referensi ini akan jarang digunakan, tetapi tetap penting untuk diketahui bahwa PHP menyediakan fitur ini.

Perhatikan contoh kode PHP berikut ini:

```
<?php
$a = 20;
$b = $a;
echo "\$a = $a, \$b = $b";
echo "<br />";
// hasil proses: $a = 20, $b = 20
$a = $a + 5;
echo "\$a = $a, \$b = $b";
echo "<br />";
// hasil proses: $a = 25, $b = 20
$b = $b + 10;
echo "\$a = $a, \$b = $b";
// hasil proses: $a = 25, $b = 30
?>
```

Dalam kode program diatas, saya membuat 2 buah variabel, yaitu **\$a** dan **\$b**. Variabel **\$a** saya input dengan nilai **20**, sedangkan variabel **\$b** **men-copy nilai** dari variabel **\$a**. Selanjutnya saya tampilkan kedua variabel tersebut menggunakan perintah echo.

Pada baris ke-9 saya menambahkan nilai variabel **\$a** dengan 5, lalu menampilkan hasil kedua variabel tersebut. Selanjutnya pada baris ke-14 saya menambahkan **\$b** dengan 10, lalu menampilkan hasilnya

Kata kunci disini adalah, variabel **\$b** hanya men-copy nilai yang ada pada variabel **\$a**, sehingga kedua variabel memiliki nilai sendiri-sendiri dan terpisah, seperti yang terlihat dari hasil **echo**.

Bagaimana jika yang saya inginkan adalah: karena variabel **\$a=\$b**, maka ketika saya merubah nilai salah satu variabel, nilai pada variabel yang lain juga ikut berubah. Fitur inilah yang bisa didapatkan dengan **Assignment by Reference**.

Assignment by Reference dalam PHP menggunakan operator "**= &**".

Perhatikan contoh kode PHP berikut yang sama persis dengan contoh sebelumnya, namun saya mengganti baris **\$b = \$a**, menjadi **\$b = &\$a**:

```
<?php
$a = 20;
$b = &$a;
echo "\$a = $a, \$b = $b";
echo "<br />";
// hasil proses: $a = 20, $b = 20
$a = $a + 5;
echo "\$a = $a, \$b = $b";
echo "<br />";
// hasil proses: $a = 25, $b = 25
$b = $b + 10;
```

```
echo "\$a = $a, \$b = $b";  
// hasil proses: $a = 35, $b = 35  
?>
```

Seperti yang dapat dilihat, bahwa sekarang kedua variabel (**\$a** dan **\$b**), seolah-olah *saling terikat*, sehingga ketika sebuah variabel diubah nilainya, variabel yang lain juga ikut berubah.

2.4.9.5 PERBEDAAN ASSIGNMENT BY VALUE, DENGAN ASSIGNMENT BY REFERENCE

Agar lebih jelas, berikut adalah Perbedaan Proses **Assignment By Value**, dengan **Assignment By Reference**. Didalam bahasa pemrograman (dan juga PHP), sebuah nilai dari **variabel** di simpan pada sebuah alamat tertentu di memory komputer. Alamat memory inilah yang dimaksud dengan **referensi**.

Misalkan variabel **\$a** memiliki nilai **20**, dan berada pada lokasi memory **1013**, ketika saya membuat kode program **\$b=\$a**, maka *nilai* (atau **value**) dari variabel **\$a** di-copy ke dalam variabel **\$b** yang mungkin saja akan berada pada lokasi memory **1014**. Sehingga saat ini ada 2 buah lokasi memori untuk menampung nilai dari masing-masing variabel, **\$a** pada lokasi **1013** dengan nilai **20**, dan **\$b** pada lokasi **1014** dengan nilai **20** (*dicopy dari nilai \$a*).

Ketika saya menambahkan variabel **\$a** dengan operasi **\$a = \$a + 5**, nilai pada lokasi memory **1013** akan menjadi **25**, namun karena *memiliki lokasi memory yang berbeda*, nilai pada variabel **\$b** akan tetap **20**. Inilah proses yang terjadi ketika menggunakan **Assignment By Value**.

Bagaimana dengan **Assignment By Reference**?

Ketika variabel **\$a** dengan nilai **20** berada di lokasi memory **1013**, dan saya menjalankan perintah **\$b=&\$a**, maka yang terjadi adalah, alamat lokasi memory (referensi) untuk variabel **\$b** dicopy dari nilai variabel **\$a**, sehingga kedua variabel memiliki 1 alamat memory yang sama, yaitu **1013**.

Karena alamat untuk variabel **\$a** dan **\$b** sama, maka ketika salah satu variabel mengubah nilai di alamat **1013**, maka *nilai tersebut akan berubah*. Ketika saya menambahkan nilai variabel **\$a** dengan operasi **\$a = \$a + 5**, nilai pada lokasi memory **1013** akan menjadi **25**. Dan ketika saya tampilkan nilai dari variabel **\$b**, maka PHP akan mencari nilai di lokasi memori **1013**, dan menampilkan hasilnya, yakni **25**. Kedua variabel **\$a** dan **\$b** *terikat dengan 1 lokasi memory yang sama*.

2.4.10 OPERATOR BITWISE

Operator **bitwise (Bitwise Operators)** adalah operator khusus yang disediakan PHP untuk menangani proses logika untuk **bilangan biner**. **Bilangan biner** atau **binary** adalah jenis bilangan yang hanya terdiri dari 2 jenis angka, yakni 0 dan 1. Jika **operand** yang digunakan untuk operator ini bukan **bilangan biner**, maka akan *dikonversi secara otomatis* oleh PHP menjadi bilangan **biner**.

Dalam penerapannya, operator **bitwise** tidak terlalu sering digunakan, kecuali anda membuah program yang langsung berkaitan dengan pemrosesan bilangan biner.

Dalam tutorial ini saya berasumsi anda telah mengetahui cara penulisan, dan perhitungan bilangan biner, jika belum silahkan mempelajarinya terlebih dahulu.

2.4.10.1 JENIS-JENIS OPERATOR BITWISE PHP

PHP mendukung 6 jenis operator **bitwise**. Daftar lengkapnya dapat dilihat pada tabel dibawah ini:

Contoh	Nama	Hasil
$\$a \& \b	And	1 Jika kedua bit bernilai 1.
$\$a \b	Or (inclusive or)	1 jika salah satu bit bernilai 1.
$\$a \wedge \b	Xor (exclusive or)	1 jika salah satu bit bernilai 1, tapi bukan keduanya
$\sim \$a$	Not	Bit 0 menjadi 1, dan bit 1 menjadi 0.
$\$a \ll \b	Shift left	Menggeser sebanyak \$b bit ke kiri (setiap 1 kali pergeseran = kelipatan 2)
$\$a \gg \b	Shift right	Menggeser sebanyak \$b bit ke kanan (setiap 1 kali pergeseran = bagi 2)

2.4.10.2 CARA PENGGUNAAN OPERATOR BITWISE DALAM PHP

Seluruh operator **bitwise** tersebut diproses dalam bentuk **biner**. Berikut contoh cara penggunaan operator **bitwise** dalam PHP:

```
<?php
$a=0b10110101;
$b=0b01101100;
echo "$a = 10110101 = $a"; echo "<br />";
echo "$b = 01101100 = $b"; echo "<br />";
echo "<br />";

echo "==Hasil Bitwise=="; echo "<br />";

$hasil = $a & $b;
echo "1. $a & $b = $hasil"; echo "<br />";

$hasil = $a | $b;
echo "2. $a | $b = $hasil"; echo "<br />";

$hasil = $a ^ $b;
echo "3. $a ^ $b = $hasil"; echo "<br />";
```

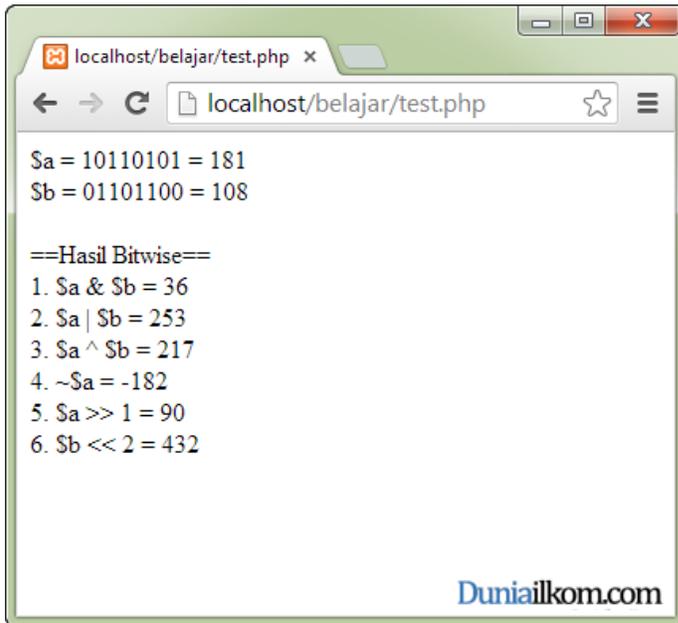
```

echo "4. ~$a = " . ~$a; echo "<br />";

$hasil = $a >> 1;
echo "5. $a >> 1 = $hasil"; echo "<br />";

$hasil = $b << 2;
echo "6. $b << 2 = $hasil"; echo "<br />";
?>

```



Dalam contoh kode PHP diatas, saya mendefenisikan 2 variabel yakni **\$a** dan **\$b**, dan memberikan nilai awal berupa angka **biner** ke dalam kedua variabel tersebut (cara penulisan angka biner telah kita bahas pada tutorial [Tutorial PHP: Mengenal Tipe Data Integer dan Cara Penulisan Integer dalam PHP](#)). Variabel **\$a** berisi angka biner: **10110101**, yang nilai desimalnya adalah **181**, dan variabel **\$b** berisi angka biner: **01101100**, yang nilai desimalnya adalah **108**.

Pada contoh 1, saya melakukan operasi **&** terhadap kedua variabel. Operasi **bitwise "and"** ini akan memproses **bit per bit** dari kedua variabel, jika kedua **bit** sama-sama 1, maka hasilnya juga 1, selain kondisi tersebut, nilai akhirnya adalah 0. Berikut perhitungan **bitwise "and"**:

```

$a =      10110101
$b =      01101100
-----
$a & $b = 00100100 = 36 (desimal)

```

Dan dari hasil ECHO, terlihat bahwa hasilnya adalah 36 (dalam bentuk DESIMAL).

1, maka ini adalah instruksi kepada PHP bahwa hasilnya akan negatif, dan hasil negatif di flip (dinegatifkan kembali), lalu dikurang 1, sehingga menjadi **-182** dalam desimal.

Contoh ke 5, adalah operator **Shift right** dimana PHP akan menggeser variabel **\$a** ke kanan sebanyak 1 tempat. Berikut proses yang terjadi:

```
$a      = 10110101 = 181
$a >> 1 = 1011010  = 90 (desimal)
```

Operator **shift right** menggeser nilai biner variabel **\$a** ke **arah kanan**, dan digit paling kanan akan dihapus. Operator **shift right** ini akan menghasilkan nilai asal / 2. Dalam contoh, hasilnya adalah 180/2 = 90 (dibulatkan). Setiap penggeseran 1 tempat ke kanan akan membagi 2 nilai asal. Contoh ke 6 adalah operator **Shift Left**, dimana PHP menggeser nilai variabel **\$b** sebanyak 2 digit **ke kiri**. Berikut proses yang terjadi:

```
$b      = 01101100 = 108
$b << 2 = 0110110000 = 432 (desimal)
```

Ketika hasil pergeseran ke kanan, digit paling kiri akan diisi dengan nilai 0. Setiap penggeseran 1 tempat ke kiri akan mengkali 2 nilai asal.

2.4.11 OPERATOR GABUNGAN

Operator gabungan assignment adalah cara penulisan singkat operator dengan menggunakan sebuah operator assignment secara bersamaan dengan operator lainnya. Dalam PHP, operator gabungan ini adalah antara operator **assignment** dengan operator lain seperti operator **aritmatika**, **string**, **bitwise**, dll.

Untuk memahami operator gabungan assignment ini, perhatikan contoh berikut:

```
<?php
$a = 10;
$a = $a + 5;
$a = $a - 10;
echo "$a = $a";
echo "<br />";

// sama hasilnya dengan kode berikut:

$b = 10;
$b += 5; // sama dengan $b = $b + 5;
$b -= 10; // sama dengan $b = $b - 10;
echo "$b = $b";
?>
```

Pada baris ke.. saya menggunakan operator gabungan untuk mempersingkat penulisan **\$b = \$b + 5** menjadi **\$b += 5;**

2.4.11.1 JENIS-JENIS OPERATOR GABUNGAN (COMBINED OPERATORS)

Operator gabungan assignment bisa digunakan hampir untuk seluruh operator lainnya, berikut tabel jenis **operator gabungan assignment** dalam PHP:

Contoh	Nama Operator	Hasil
<code>\$b += 10</code>	Plus-equals	<code>\$b = \$b + 10</code>
<code>\$b -= 10</code>	Minus-equals	<code>\$b = \$b - 10</code>
<code>\$b /= 10</code>	Divide-equals	<code>\$b = \$b / 10</code>
<code>\$b *= 10</code>	Multiply-equals	<code>\$b = \$b * 10</code>
<code>\$b %= 10</code>	Modulus-equals	<code>\$b = \$b % 10</code>
<code>\$b ^= 10</code>	Bitwise-XOR-equals	<code>\$b = \$b ^ 10</code>
<code>\$b &= 10</code>	Bitwise-AND-equals	<code>\$b = \$b & 10</code>
<code>\$b = 10</code>	Bitwise-OR-equals	<code>\$b = \$b 10</code>
<code>\$b .=</code>	Concatenate-equals	<code>\$b = \$b."duniaikom"</code>

Operator gabungan ini akan menghemat penulisan kode program, namun jika anda nyaman menggunakan kode yang sedikit panjang juga tidak masalah.

2.5 CARA MENGUBAH TIPE DATA PHP (TYPE JUGGLING DAN TYPE CASTING)

2.5.1 PENGERTIAN TYPE JUGGLING DALAM PHP

PHP merupakan bahasa pemrograman yang tidak terlalu ketat dalam aturan tipe data (dimana sebuah variabel dapat diisi dengan berbagai tipe data). Hal ini memberikan kemudahan penulisan, namun juga mendatangkan permasalahan tersendiri. Terkadang PHP mengubah tipe data suatu variabel menjadi tipe data lainnya secara tidak langsung tanpa kita instruksikan.

Jika anda telah mengikuti tutorial PHP di duniaikom tentang pembahasan tipe data dan operator, maka beberapa kali kita telah membuat kode program yang "*memaksa*" suatu tipe data berfungsi sebagai tipe data lainnya.

Perhatikan kode program php berikut:

```
<?php
$a=12;
$b="9 kucing";
echo $a+$b; // 21
?>
```

Dalam kode diatas, saya mendefinisikan variabel **\$a** sebagai **integer** (angka) dan variabel **\$b** sebagai **string**. Operasi penambahan seharusnya membutuhkan 2 inputan berupa angka, namun seperti yang terlihat ketika program dijalankan, PHP dengan senang hati akan menjalankan perintah tersebut tanpa mengeluarkan error. `12+9 kucing` akan menghasilkan nilai 21.

Proses perubahan tipe data ini dikenal dengan istilah **type juggling**. **Type Juggling** dalam PHP adalah sebuah proses untuk menentukan jenis tipe data yang "*cocok*" dengan operasi saat itu, dan PHP akan menkonversinya secara langsung. PHP akan mencoba "*menebak*" dan mengubah tipe data agar disesuaikan dengan peruntukannya. Misalkan operator aritmatika seharusnya membutuhkan 2 buah inputan (atau *operand*) yang bertipe angka (baik berupa **integer** maupun **float**). Namun apabila salah satu atau kedua operand itu bukan bertipe angka, maka PHP akan mengkonversinya menjadi angka. Seperti contoh program diatas, string "`9 kucing`" akan dikonversi menjadi angka. Dan menghasilkan angka **21** (aturan pengkonversian ini akan kita bahas sesaat lagi).

Sebagai contoh lainnya, perhatikan kode PHP berikut ini:

```
<?php
$a=12;
    $b="9 kucing";
    echo $a AND $b;
?>
```

Nilai dari variabel **\$a** dan **\$b** masih sama dengan contoh saya yang pertama, namun kali ini saya membuat operator logika **AND** sebagai operasi yang akan diproses. Dan jika anda menjalankan program diatas, di browser akan tampil angka 1. Dari manakah angka 1 ini berasal?

Operator **AND** membutuhkan 2 inputan bertipe **boolean**, yakni nilai **TRUE** atau **FALSE**. Namun karena saya menggunakan tipe integer "`12`" dan tipe string "`9 kucing`", maka kedua operand ini akan dikonversi menjadi **TRUE**. Karena **TRUE AND TRUE** adalah **TRUE**, maka hasil **\$a AND \$b** pada contoh program diatas adalah **TRUE**.

Namun proses **type juggling** PHP belum selesai, karena perintah echo membutuhkan inputan berupa **string**, bukan nilai boolean **TRUE**. Dan PHP akan mengkonversi boolean **TRUE** menjadi **string "1"**.

Pemahaman tentang **type juggling** dalam PHP ini akan membantu kita untuk menghindari error kode program karena perubahan tipe data yang tidak terdeteksi.

2.5.2 PENGERTIAN TYPE CASTING DALAM PHP

Jika **type juggling** merupakan proses “*otomatis*” oleh PHP, maka Type Casting adalah proses perubahan type data secara manual dengan menggunakan instruksi di dalam kode program. Untuk merubah sebuah type data, kita menggunakan perintah *casting* dengan cara membuat nama tipe data tujuan di dalam tanda kurung sebelum variabel yang akan diubah.

Perhatikan contoh kode program berikut:

```
<?php
    $a=12;
    $b="9 kucing";
    echo $b;           //9 kucing
    echo "<br />";
    echo (integer) $b; //9
    echo "<br />";
    echo (boolean) $b; //1
?>
```

Pada baris ke -6, saya membuat **echo (integer) \$b**, perintah ini adalah men-*casting* tipe data yang ada didalam variabel **\$b** (apapun tipe datanya) menjadi **integer**. Dan pada baris ke 8. saya meng-*casting* tipe data **\$b** menjadi **boolean**.

Jenis-jenis perintah casting yang ada dalam PHP adalah sebagai berikut:

(int), (integer) – mengubah tipe data menjadi integer

(bool), (boolean) – mengubah tipe data menjadi boolean

(float), (double), (real) – mengubah tipe data menjadi float

(string) – mengubah tipe data menjadi string

(array) – mengubah tipe data menjadi array

(object) – mengubah tipe data menjadi object

(unset) – mengubah tipe data menjadi NULL (PHP 5)

2.5.3 ATURAN KONVERSI DATA DALAM PHP

Perubahan sebuah tipe data menjadi tipe data lainnya dalam PHP memiliki aturan tersendiri. Berikut adalah aturan konversi tipe data dalam PHP

2.5.3.1 KONVERSI MENJADI INTEGER

Jika tipe data asal adalah **float**, maka perubahan menjadi **integer** akan membuang tanda desimal dari **float**. Contohnya, 3.94 akan menjadi integer 3.

Jika tipe data asal adalah **boolean**, maka nilai TRUE akan menjadi 1, dan FALSE menjadi 0.

Jika tipe data asal adalah **string**, maka string akan dipotong pada angka terakhir yang ditemukan, mulai dari awal string. Namun jika diawal string tidak terdapat angka, maka string akan dikonversi menjadi 0. Contohnya, "9 Kucing" akan menjadi integer 9, "999 kucing" akan menjadi integer 999. "kucing 99" akan menjadi 0, "14,5 kali gaji" akan menjadi integer 14 (karena desimal akan dibuang dari string)

Jika tipe data asal adalah **array** kosong (tanpa data) maka jika dikonversi menjadi integer akan menghasilkan 0, namun array dengan isi data minimal 1, akan dikonversi menjadi 1.

2.5.3.2 KONVERSI MENJADI FLOAT

Secara garis besar, konversi menjadi **float** hampir sama dengan konversi menjadi **integer**, dengan perbedaan jika tipe asal adalah **string**, maka angka **desimal** akan diperhitungkan, misalnya "14,5 kali gaji" akan dikonversi menjadi **float** 14,5

2.5.3.3 KONVERSI MENJADI BOOLEAN

Jika tipe data asal adalah **integer**, maka angka 0 akan dikonversi menjadi FALSE, selain itu, akan dikonversi menjadi TRUE. Contohnya 0 -> FALSE, 1 -> TRUE, -1 -> TRUE. Jika tipe data asal adalah **float**, maka angka 0.0 akan dikonversi menjadi FALSE, selain itu, akan dikonversi menjadi TRUE. Contohnya 0.0 -> FALSE, 1,9 -> TRUE, -1,6 -> TRUE. Jika tipe data asal adalah **string**, maka string "" (tanpa karakter) akan dikonversi menjadi FALSE, string "0" (string dengan karakter nol) akan dikonversi menjadi FALSE, selain itu akan dikonversi menjadi TRUE. Contohnya: "" -> FALSE, " " -> TRUE (karakter spasi), "0" -> FALSE. Jika tipe data asal adalah **array**, maka array kosong (tanpa data) akan dianggap FALSE, selain itu array akan dikonversi menjadi TRUE.

Sebagai penutup, berikut adalah contoh-contoh konversi tipe data didalam PHP:

```
<!DOCTYPE html>
<html>
<head>
<title>Belajar Cara Konversi Tipe Data PHP</title>
</head>
<body>
<?php
// Konversi menjadi Integer
var_dump((int) 3.45); // 3
echo "<br />";
var_dump((int) "3.45"); // 3 (string 3.45)
echo "<br />";
var_dump((int) "9 Naga"); //9
echo "<br />";
var_dump((int) "Naga Bonar"); // 0
```

```

echo "<br />";
var_dump((int) "Wiro Sableng 212"); //0
echo "<br />";
var_dump((int) FALSE); // 0
echo "<br />";
var_dump((int) "1FALSE"); //1
echo "<br />";
var_dump((int) array()); // 0
echo "<br />";
var_dump((int) array("data")); //1
echo "<br />";
// Konversi menjadi Float
var_dump((float) 3); // 3
echo "<br />";
var_dump((float) "3.45"); // 3.45
echo "<br />";
var_dump((float) "9 Naga"); //9
echo "<br />";
var_dump((float) "Naga Bonar"); // 0
echo "<br />";
var_dump((float) "Wiro Sableng 212"); //0
echo "<br />";
var_dump((float) FALSE); // 0
echo "<br />";
var_dump((float) "1FALSE"); //1
echo "<br />";
var_dump((float) array()); // 0
echo "<br />";
var_dump((float) array("data")); //1
echo "<br />";
// Konversi menjadi Boolean
var_dump((bool) 3); // TRUE
echo "<br />";
var_dump((bool) 0); // FALSE
echo "<br />";
var_dump((bool) -1); // TRUE
echo "<br />";
var_dump((bool) ""); // FALSE
echo "<br />";
var_dump((bool) " "); //TRUE
echo "<br />";
var_dump((bool) "0"); // FALSE
echo "<br />";
var_dump((bool) "FALSE");// TRUE (!) - karena string
echo "<br />";
var_dump((bool) array()); // FALSE
echo "<br />";
var_dump((bool) array("data")); // TRUE
echo "<br />";
?>
</body>
</html>

```

Dalam program diatas, saya membuat beberapa contoh konversi di dalam PHP.

3 STRUKTUR PEMROGRAMAN

3.1 LOGIKA IF

Pengertian Struktur IF dalam bahasa pemrograman adalah sebuah struktur logika untuk membuat percabangan alur program. Secara sederhananya, dengan menggunakan **struktur IF** kita dapat mengatur apakah sebuah perintah akan dijalankan atau tidak tergantung kepada kondisinya. Sebagai contoh kita ingin membuat program sederhana, jika nama user adalah “**Andi**”, maka tampilkan kata “*Selamat Datang, Andi*”. Berikut adalah penulisannya di dalam PHP:

```
<?php
    $nama="Andi";
    if ($nama=="Andi")
        echo "Selamat datang Andi, di duniaikom...";
?>
```

Jika anda jalankan kode program diatas, maka di web browser akan tampil “*Selamat datang Andi, di duniaikom...*”, namun apabila anda mengganti kode program diatas menjadi:

```
<?php
    $nama="Joni";
    if ($nama=="Andi")
        echo "Selamat datang Andi, di duniaikom...";
?>
```

Maka tidak akan ada tampilan apa-apa di dalam web browser. **Struktur logika IF** setidaknya membutuhkan 2 inputan, yaitu **ekspresi logika (*expression*)** dimana berisi kondisi yang harus dipenuhi, dan **perintah yang akan dijalankan (*statement*)** jika kondisi logika tersebut terpenuhi.

Berikut adalah struktur dasar penulisan alur **logika IF** dalam PHP:

```
if (expression)
statement
```

Expression dalam hal ini adalah kondisi yang harus dipenuhi agar *statement* dapat dijalankan. Hasil dari *expression* harus tipe **boolean**. Selama hasil *expression* bernilai **TRUE**, maka *statement* akan dijalankan, namun jika nilainya **FALSE**, maka *statement* tidak akan dijalankan.

Dalam pembuatan program, biasanya digunakan operasi perbandingan sebagai *expression*. Pada contoh kita pertama, *expression* kita adalah **\$nama=="Andi"**, yang bermaksud bahwa jika isi variabel *\$nama* sama dengan “*Andi*” maka jalankan perintah *echo*.

Namun *expression* IF ini tidak harus berupa operasi perbandingan, namun bisa berupa variabel, yang selama hasilnya adalah **TRUE**, maka *statement* akan dijalankan. Perhatikan contoh berikut ini:

```
<?php
    if (TRUE)
        echo "Selamat datang Andi, di duniaikom...";
?>
```

Jika anda menjalankan program tersebut, kalimat “*Selamat datang Andi, di duniaikom...*” akan selalu tampil di web browser, karena kondisi IF akan selalu terpenuhi.

Hasil *expression* harus bertipe **boolean**, namun dengan prinsip PHP yang menggunakan **type juggling** (dimana sebuah tipe data akan dikonversi tergantung situasinya), maka *expression* dalam percabangan IF ini bisa menggunakan tipe data selain *boolean*. Jadi anda bisa menulis seperti berikut:

```
<?php
    if (9)
        echo "Selamat datang Andi, di duniaikom...";
?>
```

Perintah **echo** akan dijalankan, karena *integer* 9, akan dikonversi menjadi **TRUE**. Lebih lanjut tentang konversi tipe data ini telah kita bahas pada [Tutorial Belajar PHP: Cara Mengubah Tipe Data PHP \(Type Juggling dan Type Casting\)](#)

3.1.1 ATURAN PENULISAN STRUKTUR IF DALAM PHP

Penulisan sederhana dari **struktur if** adalah sebagai berikut:

```
<?php
if (expression)
statement;
?>
```

Expression ditulis di dalam tanda kurung, dan tidak diikuti dengan titik koma(;). Apabila *statement* yang ingin dijalankan terdiri dari 2 baris atau lebih, kita harus memberikan tanda kurung kurawal untuk menandai *statement* yang berhubungan dengan kondisi IF. Berikut contoh strukturnya:

```
<?php
if (expression)
{
    statement1;
    statement1;
}
?>
```

Tanda kurung kurawal menandakan blok perintah yang dijalankan jika *expression* bernilai true. Kita juga bisa membuat beberapa logika IF sekaligus untuk berbagai situasi:

```

<?php
if (expression1)
{
statement1;
statement2;
}
if (expression2)
{
statement3;
statement4;
}
?>

```

Untuk kasus yang lebih spesifik, kita bisa membuat struktur IF didalam IF, atau dikenal dengan **nested IF**, seperti contoh berikut:

```

if (expression)
{
statement1;
if (expression)
{
statement1;
}
}
?>

```

Seberapa banyak kondisi IF didalam IF (*nested*) tidak dibatasi dalam PHP, namun perlu diperhatikan penggunaan tanda kurung kurawal sebagai penanda bagian dari IF. Jika anda membuat **struktur IF** yang kompleks, tanda kurung kurawal ini akan membuat bingung jika tidak dikelola dengan benar. Kesalahan penutupan kurung kurawal akan membuat program tidak berjalan sesuai dengan keinginan.

3.1.2 ALTERNATIF PENULISAN STRUKTUR LOGIKA IF

Selain menggunakan tanda *kurung kurawal* sebagai tanda awal dan akhir IF, PHP menyediakan cara penulisan lain untuk menandai akhir perintah IF, yaitu diawali dengan tanda titik dua (:) dengan diakhiri dengan **endif**. Berikut adalah format dasar penulisan IF:

```

<?php
if (expression) :
    statement1;
    statement1;
endif
?>

```

Perbedaan mendasar tentang cara penulisan ini ada di tanda titik dua (:) setelah penulisan expression, dan kata kunci **endif** di akhir statement. Anda bebas menggunakan format penulisan logika IF yang disediakan. Beberapa programmer menggunakan alternatif penulisan IF dengan **endif** ini, karena dianggap lebih rapi.

3.2 LOGIKA ELSE

Jika **Struktur IF** digunakan untuk percabangan alur program dengan 1 pilihan saja, maka dengan struktur **ELSE** kita dapat membuat percabangan kedua, yakni percabangan ketika kondisi **IF** tidak terpenuhi, atau expressi **IF** menghasilkan nilai **FALSE**.

Berikut adalah contoh penggunaan logika **ELSE** dalam **PHP**:

```
<?php
$nama="Andi";
if ($nama=="Andi")
    echo "Selamat Datang Andi...";
else
    echo "Selamat Datang di duniailkom";
?>
```

Contoh kode program diatas, hampir sama dengan contoh kita pada tutorial tentang **IF** sebelumnya. Namun kali ini saya menambahkan **percabangan ELSE**. Jika anda menjalankan kode program diatas, maka di dalam web browser akan tampil *"Selamat Datang Andi..."*, karena kondisi **IF** terpenuhi. Perintah **ELSE** hanya akan dijalankan jika kondisi **\$nama** bukan berisi *"ANDI"*.

Jika saya mengubah kode **PHP** diatas menjadi

```
<?php
$nama="Budi";
if ($nama=="Andi")
    echo "Selamat Datang Andi...";
else
    echo "Selamat Datang di duniailkom";
?>
```

Maka sekarang di web browser akan tampil *"Selamat Datang di duniailkom"*. Hal ini terjadi karena kondisi **if (\$nama=="Andi")** tidak terpenuhi dan menghasilkan **FALSE**, sehingga perintah di bagian **ELSE**-lah yang akan dieksekusi.

3.2.1 ATURAN PENULISAN STRUKTUR IF-ELSE DALAM PHP

Penulisan sederhana dari struktur **IF-ELSE** adalah sebagai berikut:

```
<?php
if (expression)
    statement1;
else
    statement2;
?>
```

Statement1 akan dijalankan hanya jika *expression* bernilai **TRUE** (kondisi *expression* terpenuhi). Namun apabila kondisi *expression* tidak terpenuhi (bernilai **FALSE**), maka *statement2* lah yang akan dijalankan.

Jika struktur logika **IF-ELSE** terdiri dari beberapa baris, maka kita harus menambahkan penanda kurung kurawal untuk menandai awal dan akhir statement. Penanda ini dibutuhkan untuk membatasi blok perintah mana yang akan dijalankan ketika *expression* **TRUE**, dan blok perintah mana yang akan dijalankan jika *expression* **FALSE**. Berikut adalah penulisan dasar struktur **IF-ELSE** dengan pembatasan blok perintah:

```
<?php
if (expression)
{
statement1;
statement1;
}
else
{
statement2;
statement1;
}
?>
```

Penandaan *statement* ini akan menghasilkan *error* ketika kita salah atau lupa menempatkan tanda *kurung kurawal*. Perhatikan contoh kode PHP berikut ini:

```
<?php
$nama="Budi";
if ($nama=="Andi")
    echo "Selamat Datang Andi...";
    echo "Anda Memiliki 3 pesan di inbox...";
else
    echo "Maaf, anda tidak memiliki hak akses";
?>
```

Kode PHP diatas akan menghasilkan **error**, karena PHP mendeteksi ada lebih dari satu baris setelah struktur IF. Kode tersebut akan berjalan seperti yang diinginkan jika dirubah menjadi:

```
<?php
$nama="Budi";
if ($nama=="Andi")
{
    echo "Selamat Datang Andi...";
    echo "Anda Memiliki 3 pesan di inbox...";
}
else
{
    echo "Maaf, anda tidak memiliki hak akses";
}
?>
```

Pada baris terakhir, saya juga menambahkan tanda kurung kurawal sebagai penanda awal dan akhir dari **ELSE**, walaupun tanda kurung tersebut sebenarnya tidak diperlukan (karena hanya bersisi satu baris). Namun hal ini akan memudahkan kita seandainya ingin menambahkan perintah tambahan pada bagian **ELSE**.

3.2.2 CARA PENULISAN ALTERNATIF STRUKTUR ELSE

Sama seperti alternatif penulisan IF pada tutorial sebelumnya, selain menggunakan tanda kurung kurawal penanda awal dan akhir blok IF, PHP juga menyediakan cara penulisan lain untuk blok perintah ELSE, yaitu diawali dengan **tanda titik dua** (:) dan diakhiri dengan **endif**. Berikut adalah format dasar penulisan IF:

```
<?php
if (expression) :
    statement1;
    statement2;
else:
    statement3;
endif
?>
```

Perbedaan mendasar tentang cara penulisan ini ada di setelah penulisan expression dimana dibutuhkan **tanda titik dua** (:), dan di akhir statement dengan kata kunci **endif**. Anda bebas menggunakan format penulisan logika IF yang disediakan. Beberapa programmer memilih alternatif penulisan IF dengan **endif** karena dianggap lebih rapi.

3.3 LOGIKA ELSE IF

Struktur ELSE-IF merupakan percabangan logika lanjutan dari IF. Dengan **ELSE-IF** kita bisa membuat kode program yang akan menyeleksi berbagai kemungkinan yang bisa terjadi. Berikut adalah contoh penggunaan **ELSE-IF** dalam PHP:

```
<?php
$a=15;
$b=8;

if ($a > $b)
{
    echo "a lebih besar daripada b";
}
elseif ($a == $b)
{
    echo "a sama besar dengan b";
}
else
{
    echo "a lebih kecil daripada b";
}
?>
```

Dalam kode program diatas, saya membuat program sederhana untuk membandingkan 2 angka. IF pertama akan melakukan pengecekan apakah **\$a > \$b**, jika hasilnya adalah **FALSE**, maka masuk ke IF

kedua (ditulis dengan **elseif**) apakah **\$a == \$b**, dan jika hasilnya adalah **FALSE**, maka dapat dipastikan **\$a < \$b**.

Jika anda bertanya apa perbedaan IF dengan **ELSEIF**, maka jawabanya terletak di efisiensi pemrosesan.

Contoh diatas bisa juga kita buat tanpa menggunakan **ELSEIF** seperti berikut ini:

```
<?php
$a=15;
$b=8;

if ($a > $b)
{
    echo "a lebih besar daripada b";
}
if ($a == $b)
{
    echo "a sama besar dengan b";
}
else
{
    echo "a lebih kecil daripada b";
}
?>
```

Perhatikan kode program pada baris ke-9, saya mengganti **ELSEIF** menjadi IF.

Perbedaannya adalah, untuk contoh kode PHP kita tanpa IF ini, seluruh kondisi akan dijalankan, walaupun sebenarnya tidak perlu. Jika **\$a=15** dan **\$b=8**, maka kondisi IF pertama akan terpenuhi (**\$a > \$b**), dan kita ingin program PHP keluar dari IF. Namun karena perintah selanjutnya adalah IF, maka PHP akan tetap memeriksa apakah (**\$a == \$b**). Lain halnya jika kita menggunakan perintah **ELSEIF**, maka ketika sebuah kondisi telah dipenuhi, PHP tidak perlu melakukan pengecekan terhadap kondisi IF lainnya.

3.3.1 ATURAN PENULISAN STRUKTUR ELSE-IF DALAM PHP

Dalam PHP, kita bisa menuliskan struktur **ELSE-IF** dengan **elseif**, atau **else if** (dipisahkan dengan spasi).

Kedua bentuk ini dianggap sama. Format dasar penulisan ELSE-IF adalah sebagai berikut:

```
<?php
if (expression)
{
    statement1;
}
elseif
{
    statement2;
}
else
{
    statement3;
}
```

```
?>
```

Seberapa banyak struktur **ELSE-IF** di dalam kode program tidak dibatasi, namun jika anda ada dalam situasi yang membutuhkan percabangan **ELSE IF** yang lebih dari 5, mungkin anda bisa memecah nya menjadi bagian-bagian kecil agar memudahkan alur logika program.

3.3.2 CARA PENULISAN ALTERNATIF STRUKTUR ELSE-IF

Selain menggunakan tanda kurung kurawal sebagai tanda awal dan akhir **ELSE-IF**, PHP menyediakan cara penulisan alternatif. Berikut format dasar penulisannya:

```
<?php
if (expression) :
    statement1;
    statement2;
elseif (expression):
    statement3;
else
    statement4;
endif
?>
```

Namun untuk cara penulisan ini, kita tidak bisa memisahkan penulisan **ELSE-IF** menjadi “**else if**”, tetapi harus ditulis menyatu menjadi “**elseif**”.

```
<?php
$a=15;
$b=8;

if ($a > $b):
    echo "a lebih besar daripada b";
else if ($a == $b): // akan menghasilkan error
    echo "a sama besar dengan b";
else:
    echo "a lebih kecil daripada b";
endif;
?>
```

Kode program diatas baru berhasil dieksekusi jika diubah menjadi:

```
<?php
$a=15;
$b=8;

if ($a > $b):
    echo "a lebih besar daripada b";
elseif ($a == $b):
    echo "a sama besar dengan b";
else:
    echo "a lebih kecil daripada b";
endif;
?>
```

Struktur IF-ELSE-IF ini merupakan salah satu struktur terpenting dalam pemrograman, dengan struktur IF kita bisa membuat alur percabangan program tergantung dengan situasi yang dihadapi.

3.4 LOGIKA SWITCH

Struktur logika switch adalah sebuah stuktur percabangan yang akan memeriksa suatu *variabel*, lalu menjalankan perintah-perintah yang sesuai dengan kondisi yang mungkin terjadi untuk variabel tersebut. Struktur *switch* ini mirip dengan struktur IF yang ditulis berulang.

Katakan kita ingin membuat sebuah program yang akan menampilkan kata dari angka 0-5, sehingga terdapat 6 kemungkinan yang terjadi. Jika menggunakan struktur IF, maka kita akan membutuhkan 6 perulangan sebagai berikut:

```
<?php
$a=3;
if ($a=="0") {
    echo "Angka Nol";
}
elseif ($a=="1") {
    echo "Angka Satu";
}
elseif ($a=="2") {
    echo "Angka Dua";
}
elseif ($a=="3") {
    echo "Angka Tiga";
}
elseif ($a=="4") {
    echo "Angka Empat";
}
elseif ($a=="5") {
    echo "Angka Lima";
}
else
    echo "Angka diluar jangkauan";
?>
```

Tidak ada yang salah dari kode program tersebut, namun jika kita menggunakan **switch**, kode tersebut dapat ditulis menjadi:

```
<?php
$a=3;
switch ($a)
{
case 0 :
    echo "Angka Nol";
    break;
case 1 :
    echo "Angka Satu";
    break;
case 2 :
    echo "Angka Dua";
```

```

        break;
    case 3 :
        echo "Angka Tiga";
        break;
    case 4 :
        echo "Angka Empat";
        break;
    case 5 :
        echo "Angka Lima";
        break;
    default :
        echo "Angka diluar jangkauan";
        break;
}
?>

```

Kedua kode program akan menghasilkan output yang sama, namun untuk kondisi logika yang diuji merupakan kondisi sederhana, penulisan dengan switch lebih disarankan dibandingkan IF.

3.4.1 ATURAN PENULISAN STRUKTUR SWITCH DALAM PHP

Seperti yang terlihat dalam contoh sebelumnya, struktur **switch** terdiri dari beberapa bagian, berikut format dasar penulisan **switch** dalam PHP:

```

switch ($var)
{
case value1:
statement1;
break;
case value2:
statement2;
break;
}

```

Setelah kata kunci **switch**, kita harus mencantumkan *variabel* yang akan diperiksa nilainya didalam tanda kurung, lalu memulai block **switch** dengan *kurung kurawal*.

Tiap kondisi yang mungkin terjadi dicantumkan setelah kata kunci **case**, lalu diikuti dengan nilai yang akan dibandingkan dengan nilai *variabel switch*. Jika kondisi sesuai, maka baris program *statement* akan dijalankan. Kata kunci **break** digunakan untuk keluar dari *switch*, sehingga PHP tidak perlu memeriksa *case* berikutnya.

Alur program untuk **switch** akan dieksekusi dari baris pertama sampai terakhir. Kata kunci **break** memegang peranan penting untuk menghentikan **switch**.

Perhatikan contoh kode PHP berikut:

```

<?php
$a=1;
switch ($a)

```

```

{
case 0:
    echo "Angka Nol ";
case 1 :
    echo "Angka Satu ";
case 2 :
    echo "Angka Dua ";
case 3 :
    echo "Angka Tiga ";
}
?>

```

Program diatas akan memeriksa nilai dari **\$a**, dan memberikan output tergantung kepada nilai **\$a** tersebut. Jika sekilas dilihat, maka keluaran program adalah: *"Angka Satu"* sesuai dengan nilai variabel **\$a**. Akan tetapi, jika anda menjalankan program diatas, PHP akan memberikan output berupa:

```

Angka Satu Angka Dua Angka Tiga

```

Apa yang terjadi? Hal ini terkait dengan bagaimana PHP menjalankan proses **switch**.

Ketika program dijalankan, PHP pertama kali akan memeriksa **case 0**, yaitu apakah **\$a** sama dengan **0**, jika tidak, PHP akan lanjut ke **case 1**, dan memeriksa apakah **\$a** sama dengan **1**. Jika iya, maka PHP akan menjalankan **echo "Angka Satu"**, beserta seluruh perintah program pada case-case dibawahnya. Hal ini mungkin terkesan aneh, namun adakalanya proses seperti inilah yang dibutuhkan.

Namun, untuk kasus diatas, kita ingin menginstruksikan kepada PHP bahwa setelah case ditemukan, maka *switch* harus berhenti. Untuk instruksi ini, kita harus menggunakan kata kunci **break**. Instruksi **break** memberitahu PHP untuk segera keluar dari **switch**, dan tidak menjalankan case lainnya.

Berikut adalah kode program *switch* kita setelah ditambahkan keyword **break**:

```

<?php
$a=1;
switch ($a)
{
case 0:
    echo "Angka Nol ";
    break;
case 1 :
    echo "Angka Satu ";
    break;
case 2 :
    echo "Angka Dua ";
    break;
case 3 :
    echo "Angka Tiga ";
    break;
}
?>

```

Selain kata kunci `break`, PHP menyediakan kata kunci **default** untuk alur **switch**. Kata kunci ini berfungsi seperti **ELSE** di dalam struktur **IF**, yakni kondisi dimana seluruh *case* untuk switch tidak ada yang cocok. Kata kunci **default** ini diletakkan di akhir dari switch.

Untuk contoh kita diatas, saya akan menambahkan bagian **default** sebagai perintah yang akan dijalankan jika nilai dari variabel `$a` diluar dari angka 0-5. Berikut kode PHP nya:

```
<?php
$a=9;
switch ($a)
{
case 0:
    echo "Angka Nol ";
    break;
case 1 :
    echo "Angka Satu ";
    break;
case 2 :
    echo "Angka Dua ";
    break;
case 3 :
    echo "Angka Tiga ";
    break;
default :
    echo "Angka diluar jangkauan";
    break;
}
?>
```

PHP membolehkan kita menjalankan satu statement saja untuk case yang berlainan, seperti contoh kode PHP berikut ini:

```
<?php
$a=3;
switch ($a)
{
case 0 :
case 1 :
case 2 :
case 3 :
    echo "Angka berada di dalam range 0-3";
    break;
case 4 :
case 5 :
case 6 :
case 7 :
    echo "Angka berada di dalam range 4-7";
    break;
default :
    echo "Angka diluar jangkauan";
    break;
}
?>
```

Didalam kode diatas, saya menyatukan beberapa case ke dalam 1 statement. Penulisan case untuk struktur **switch** menyesuaikan dengan jenis tipe data yang akan diuji. Sampai dengan bagian ini, saya hanya menggunakan contoh case untuk variabel dengan tipe angka, namun jika anda menggunakan **switch** untuk tipe data **string**, maka kita harus menggunakan tanda kutip untuk case.

Berikut contoh kode switch PHP untuk tipe data string:

```
<?php
$a=dua;
switch ($a)
{
case "nol":
    echo "Angka 0 ";
    break;
case "satu" :
    echo "Angka 1 ";
    break;
case "dua" :
    echo "Angka 2 ";
    break;
case "tiga" :
    echo "Angka 3 ";
    break;
default :
    echo "Angka diluar jangkauan";
    break;
}
?>
```

3.4.2 PERBEDAAN ANTARA STRUKTUR IF DENGAN SWITCH

Walaupun memiliki tujuan yang hampir sama, namun struktur IF dan switch memiliki perbedaan yang mendasar. Didalam struktur **switch**, kondisi logika hanya akan diperiksa satu kali saja, yaitu pada awal perintah **switch**, dan hasilnya di dibandingkan dengan setiap **case**. Akan tetapi di dalam struktur **if**, setiap kondisi akan selalu diperiksa. Sehingga jika anda memiliki struktur percabangan yang banyak, struktur **switch** akan lebih cepat dieksekusi.

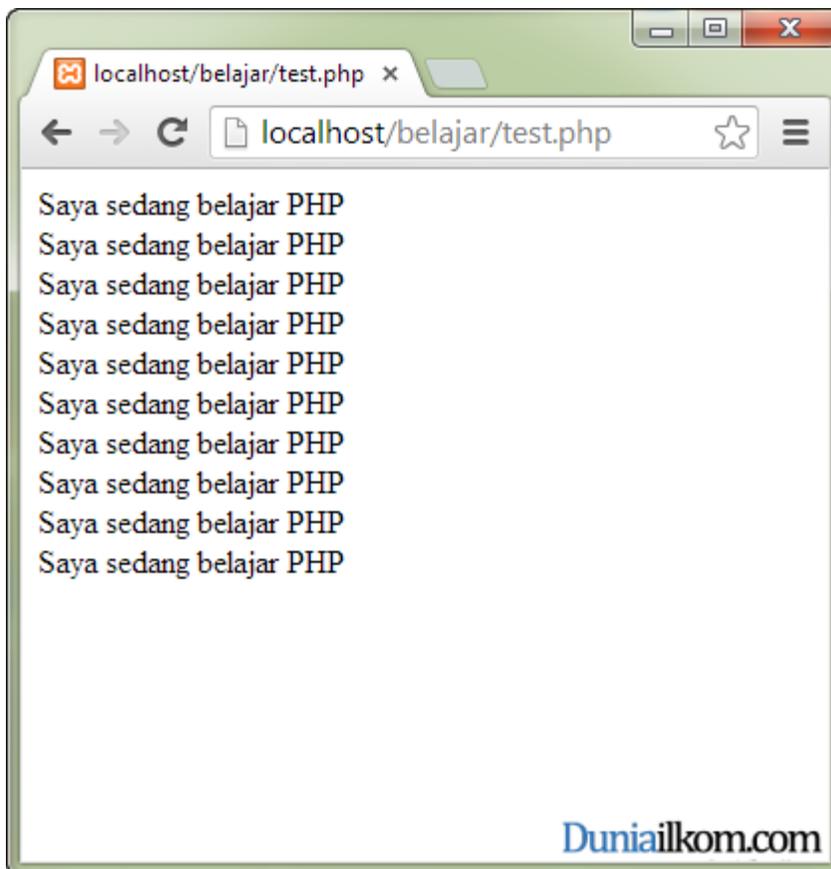
Namun disisi lain, **switch** memiliki keterbatasan dalam jenis operasi perbandingan yang dapat dilakukan. Operasi perbandingan di dalam **switch** terbatas untuk hal-hal sederhana seperti memeriksa nilai dari sebuah variabel.

Struktur **switch** tidak bisa digunakan untuk percabangan program dengan operasi yang lebih rumit seperti membandingkan 2 variabel. Kita tidak bisa menggunakan switch untuk membuat kode program menentukan nilai terbesar seperti contoh pada tutorial IF sebelum ini. Untuk kebanyakan kasus, kita akan sering menggunakan IF dibandingkan **switch**.

3.5 PERULANGAN FOR

Struktur perulangan (atau dalam *bahasa inggris* disebut dengan **loop**) adalah instruksi program yang bertujuan untuk mengulang beberapa baris perintah. Dalam merancang perulangan kode program, kita setidaknya harus mengetahui 3 komponen, yaitu *kondisi awal dari perulangan*, *perintah program yang akan diulang*, serta *kondisi akhir dimana perulangan akan berhenti*. Di dalam bahasa pemrograman, terdapat beberapa jenis instruksi perulangan, salah satunya: **struktur perulangan FOR**. Sebagai contoh sederhana untuk **perulangan for**, saya akan membuat program PHP untuk menampilkan 10 baris kalimat "Saya sedang belajar PHP". Berikut adalah kode program yang digunakan:

```
<?php
for ($i= 1; $i <= 10; $i++)
{
    echo "Saya sedang belajar PHP";
    echo "<br />";
}
?>
```



Jika anda menjalankan perintah tersebut, maka di *web browser* akan tampil sebanyak 10 kalimat. Kemampuan bahasa pemrograman untuk melakukan perulangan ini sangat praktis jika yang kita inginkan adalah mengulang beberapa perintah yang sama sebanyak beberapa kali.

3.5.1 CARA PENULISAN STRUKTUR PERULANGAN FOR DALAM PHP

Seperti yang telah saya singgung sebelumnya, untuk kondisi **perulangan for**, kita setidaknya membutuhkan 3 kondisi, yaitu di kondisi **awal perulangan**, kondisi pada **saat perulangan**, dan kondisi yang harus dipenuhi agar **perulangan berhenti**.

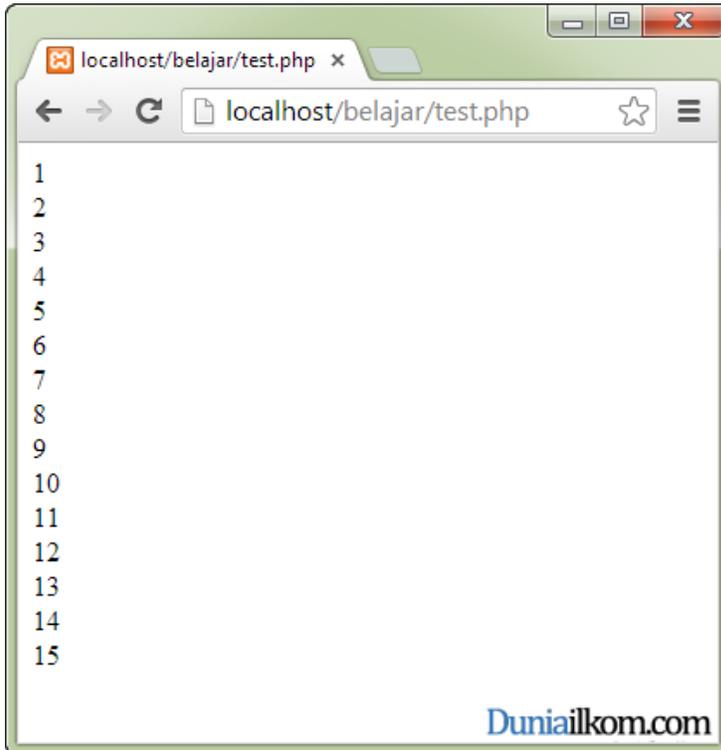
Penulisan dasar format **perulangan for** PHP adalah sebagai berikut:

```
for (start; condition; increment)
{
statement;
}
```

- **Start** adalah kondisi pada saat awal perulangan. Biasanya kondisi awal ini digunakan untuk membuat dan memberikan nilai kepada **variabel counter** yang digunakan untuk mengontrol perulangan. Misalkan, kita akan membuat variabel counter **\$i**, maka untuk kondisi start ini, kita juga harus memberikan nilai awal untuk variabel **\$i**, misalnya dengan **1**, maka **\$i=1**.
- **Condition** adalah kondisi yang harus dipenuhi agar perulangan dijalankan. Selama kondisi ini terpenuhi, maka PHP akan terus melakukan perulangan. Biasanya **variabel counter** digunakan untuk mengatur akhir perulangan. Misalkan kita ingin menghentikan perulangan jika variabel **\$i** telah mencapai nilai 20, maka pada bagian **condition** ini kita membuat perintah **\$i<=20**, yang berarti selama nilai **\$i** kurang atau sama dengan 20, terus lakukan perulangan.
- **Increment** adalah bagian yang digunakan untuk memproses variabel counter agar bisa memenuhi kondisi akhir perulangan. Biasanya, pada bagian inilah kita akan membuat kondisi dari **variabel counter**.
- **Statement** adalah bagian kode program yang akan diproses secara terus-menerus selama proses perulangan berlangsung. Untuk **statement** ini, kita membuat blok program di antar **tanda kurung kurawal** (**{** dan **}**) sebagai penanda bahwa bagian di dalam kurung kurawal inilah yang akan dikenai proses perulangan.

Sebagai contoh, kita akan membuat perulangan untuk menampilkan angka 1-15 kedalam web browser, berikut kode PHP yang digunakan:

```
<?php
for ($i= 1; $i <= 15; $i++)
{
echo $i;
echo "<br />";
}
?>
```



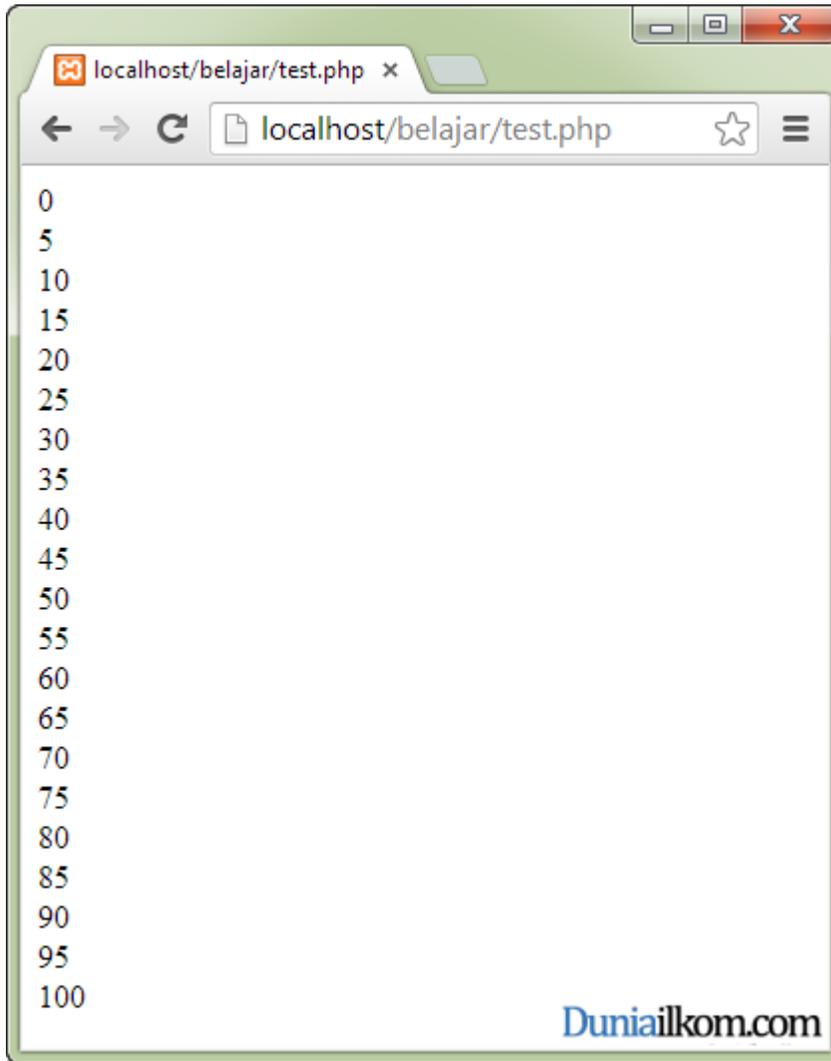
Jika anda menjalankan kode tersebut, maka di dalam web browser akan tampil urutan angka dari 1 sampai dengan 15. Sebagai kondisi awal dari perulangan tersebut adalah **$\$i = 1$** , dimana saya memberikan nilai 1 kepada variabel **$\$i$** . Variabel **$\i** inilah yang akan menjadi **counter** atau *penghitung* dari **perulangan for**. Untuk kondisi akhir, saya membuat **$\$i \leq 15$** , jadi selama variabel **$\$i$** bernilai kurang atau sama dengan 15, maka perulangan akan terus dijalankan.

Sebagai **increment**, saya membuat **$\$i++$** , dimana instruksi ini sama dengan **$\$i = \$i + 1$** . instruksi ini akan dijalankan pada setiap perulangan, sehingga dengan kata lain, setiap proses perulangan, **$\$i$** akan bertambah 1 angka. Selain berfungsi sebagai **counter**, variabel **$\$i$** juga dapat digunakan dalam proses perulangan, sehingga dengan membuat perintah **echo $\$i$** , maka dalam setiap perulangan, kita bisa menampilkan nilai **$\$i$** pada saat itu.

Sebagai contoh lain, saya ingin membuat perulangan untuk menampilkan angka 0-100, namun untuk kelipatan 5, seperti: 0.5.10..dst, sampai dengan 100.

Berikut adalah contoh kode PHPnya:

```
<?php
for ($i= 0; $i <= 100; $i=$i+5)
{
echo $i;
echo "<br />";
}
```



Perbedaan penulisan **struktur for** diatas dibandingkan contoh sebelumnya adalah pada bagian **increment**, dimana saya membuat kondisi **increment** yang menaik sebanyak 5 angka setiap perulangannya (**$\$i=\$i+5$**). Sehingga **variabel counter**, **$\$i$** akan bertambah sebanyak 5 pada setiap perulangan.

Kita juga bisa membuat perulangan dengan kondisi mundur, seperti contoh kode PHP berikut ini:

```
<?php
for ($i= 20; $i >= 1; $i--)
{
echo $i;
echo "<br />";
}
```

Di dalam kode tersebut, saya memulai nilai awal dari angka **$\$i= 20$** , membuat perulangan selama **$\$i >= 1$** , dan pada setiap perulangan, nilai **$\$i$** akan dikurangi 1 angka (**$\$i-$**). Dengan kondisi tersebut, maka variabel counter **$\$i$** akan dikurangi 1 pada setiap perulangan.

3.5.2 PENGERTIAN INFINITY LOOP

Ketika membuat kondisi akhir dari perulangan for, kita harus memperhatikan *kapan kondisi akhir tersebut dipenuhi*. Jika kondisi akhir tidak pernah terpenuhi, maka perulangan akan berjalan selamanya. Hal ini dikenal dengan **infinity loop**. Seperti yang terjadi untuk kode seperti berikut ini:

```
<?php
for ($i= 20; $i >= 1; $i++)
{
echo $i;
echo "<br />";
}
```

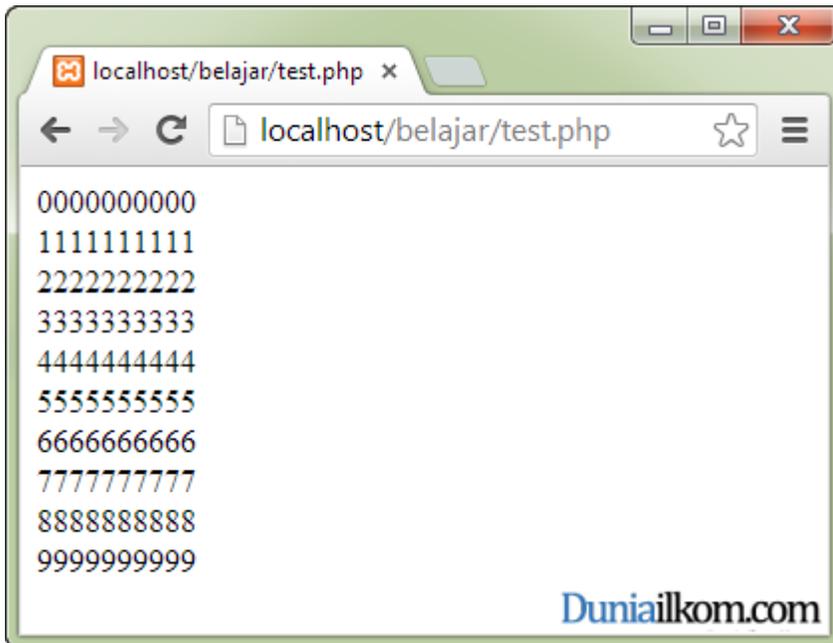
Jika anda menjalankan kode tersebut, proses perulangan akan berjalan terus menerus, sehingga untuk menghentikannya kita harus menutup paksa *web browser*. Kesalahan dari **struktur for** tersebut adalah pada *kondisi akhir dari perulangan*, dimana saya membuat **\$i >= 1**, sehingga ketika nilai awal variabel counter **\$i** adalah 20, dan dalam tiap perulangan **\$i** ditambah 1, maka nilai **\$i** akan selalu lebih besar dari 1, sehingga kondisi akhir tidak akan pernah terpenuhi, dan **\$i >= 1** akan selalu benar.

Infinity loop ini kadang diperlukan untuk kasus-kasus tertentu. Namun kebanyakan kita akan menghindari perulangan jenis ini.

3.5.3 PENGERTIAN NESTED LOOP (PERULANGAN BERSARANG)

Selain **infinity loop**, terdapat istilah lainnya yang sering digunakan dalam proses perulangan, yakni **nested loop**, atau terjemahan bebasnya: *perulangan bersarang*. **Nested loop** adalah istilah pemrograman yang berarti membuat perulangan di dalam perulangan. Perhatikan contoh program berikut:

```
<?php
for ($i=0; $i <10; $i++)
{
for ($j=0; $j <10; $j++)
{
echo $i;
}
echo "<br />";
}
```



Dalam contoh program diatas, saya membuat perulangan di dalam perulangan. Counter **\$j** digunakan untuk *perulangan dalam (inner loop)*, dan counter **\$i** digunakan di dalam *perulangan luar (outer loop)*. **Nested loop** ini biasanya digunakan dalam program yang membutuhkan pengaksesan kompleks, seperti array 2 atau 3 dimensi.

3.5.4 CARA ALTERNATIF PENULISAN PERULANGAN FOR DALAM PHP

Sama seperti **struktur if**, PHP juga memiliki alternatif perulangan tanpa menggunakan tanda **kurung kurawal** sebagai penanda blok program, dan menggantinya dengan **endfor**.

Berikut adalah contoh perulangan for dengan **endfor**:

```
<?php
for ($i= 1; $i <= 15; $i++) :
echo $i;
echo "<br />";
endfor;
?>
```

Perbedaan dengan penulisan for dengan **kurung kurawal** adalah penanda awal blok yang menggunakan **tanda titik dua (:)** dan pada akhir blok dengan perintah **endfor**.

3.6 PERULANGAN DO WHILE

Untuk situasi dimana kita membutuhkan kondisi perulangan yang tidak dapat dipastikan berapa kali perulangan akan dilakukan, maka kita tidak bisa menggunakan **perulangan for**.

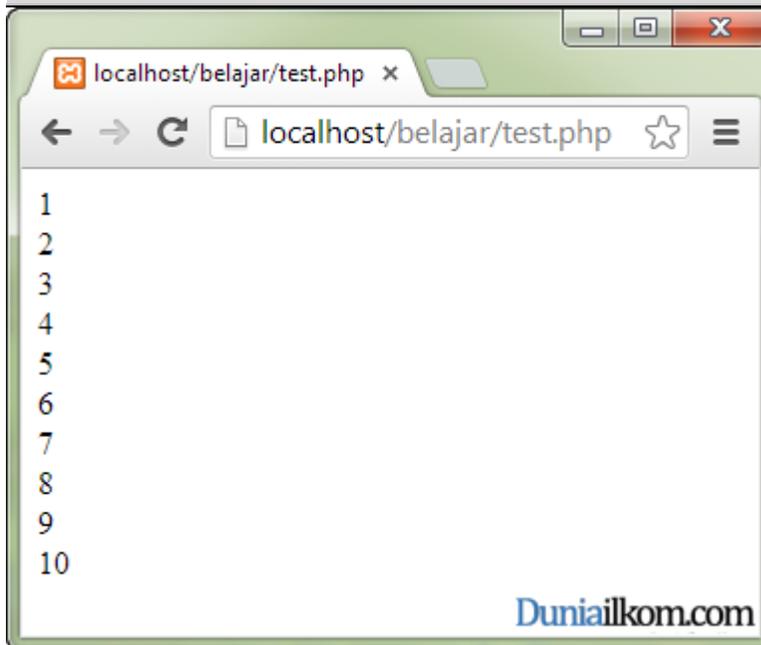
PHP (dan juga bahasa pemrograman lain) menyediakan struktur **perulangan while** untuk kondisi perulangan dimana *banyaknya perulangan tidak dapat dipastikan* pada saat penulisan program.

Misalkan kita ingin membuat program tebak angka, dimana user akan menebak 1 angka dari 1 sampai 10. Untuk kondisi ini, kita tidak dapat mengetahui berapa kali user akan '*mencoba*' untuk menebak angka tersebut. Bisa saja user mencoba sebanyak 1, 5, atau 10 kali sebelum angka tersebut berhasil diterka.

Atau misalkan kita ingin membuat program menemukan kata tertentu di dalam sebuah kalimat yang dimasukkan oleh user. Banyak kata dalam kalimat tidak bisa kita tentukan pada saat pembuatan program (karena akan diinput oleh user pada saat program berjalan), maka kita tidak bisa menentukan seberapa banyak perulangan yang harus dilakukan untuk mencari kata tersebut.

Sampai dengan tutorial ini, kita belum bisa membuat kedua program diatas karena membutuhkan beberapa fungsi PHP yang belum kita pelajari, namun sebagai contoh cara penggunaan **struktur while** sederhana, berikut adalah kode PHP untuk perulangan **while**:

```
<?php
$i=1;
while ($i <= 10)
{
    echo "$i";
    echo "<br />";
    $i=$i+1;
}
?>
```



Jika anda menjalankan kode program tersebut, maka di **web browser** akan tampil perulangan angka 1 sampai dengan 10. Dalam tutorial ini, kita akan membahas cara penulisan **struktur while** ini secara lebih dalam.

3.6.1 CARA PENULISAN STRUKTUR PERULANGAN WHILE PHP

Seperti terlihat pada contoh program sebelumnya, struktur **while** dalam PHP terdiri dari 2 bagian, yaitu *kondisi yang harus dipenuhi untuk proses perulangan*, dan *baris perintah yang akan diproses secara berulang*.

Struktur dasar **perulangan while** adalah sebagai berikut:

```
while (condition)
{
    statement;
    statement;
}
```

- **Condition** adalah kondisi yang harus dipenuhi agar perulangan berlangsung. Kondisi ini mirip seperti dalam **perulangan for**. Selama **condition** bernilai **TRUE**, maka perulangan akan terus dilakukan. **Condition** ini akan diperiksa pada tiap perulangan, dan hanya jika hasilnya **FALSE**, maka proses perulangan berhenti.
- **Statement** adalah kode program yang akan diulang. Kita bisa membuat beberapa kode program untuk menampilkan perintah seperti *echo*, atau perintah yang lebih kompleks. Namun di dalam bagian ini harus ada baris program yang digunakan sebagai *'penghenti'* perulangan. Misalkan pada bagian **condition** kita menggunakan *variabel counter \$i*, maka di bagian statement harus ada baris program yang membuat **condition** bernilai **FALSE**, atau kalau tidak proses perulangan tidak akan pernah berhenti (*infinity loop*).
- Tanda **kurung kurawal** diperlukan untuk membatasi blok program yang akan diulang. Jika statement hanya terdiri dari 1 baris, maka tanda kurung kurawal tidak diperlukan.

Sebagai pembahasan, saya akan menampilkan ulang contoh kode program sebelumnya, yakni:

```
<?php
$i=1;
while ($i <= 10)
{
echo "$i";
echo "<br />";
$i=$i+1;
}
?>
```

Pada baris ke-2 saya membuat sebuah variabel **\$i**, dan memberikan nilai 1. Variabel **\$i** inilah yang akan digunakan sebagai **counter** untuk **kondisi while**.

Setelah penulisan **while**, selanjutnya didalam tanda kurung adalah **condition** yang harus dipenuhi agar perulangan berjalan. Saya membuat kondisi (**\$i <= 10**) sebagai penanda akhir **while**, yang berarti selama variabel **\$i** bernilai kurang dari 10, maka lakukan perulangan.

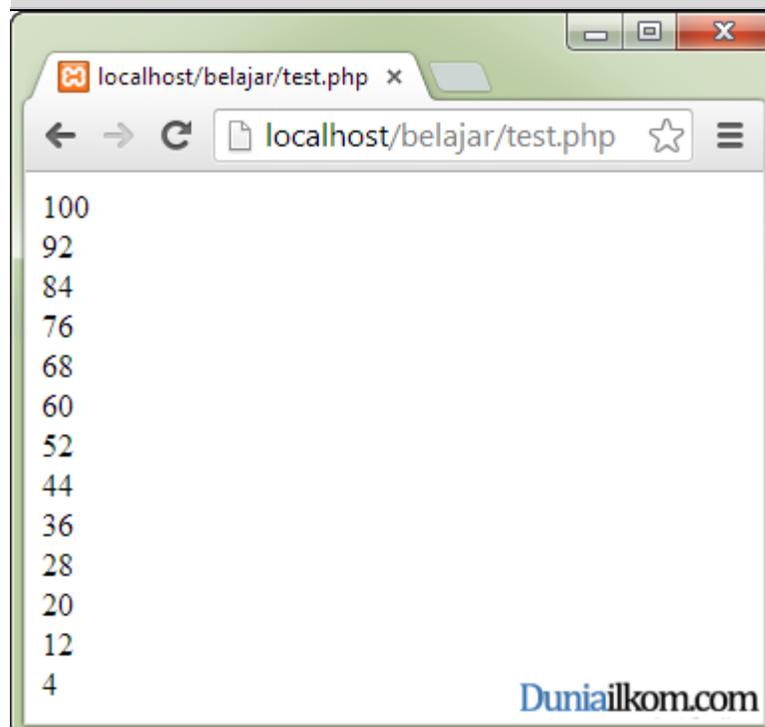
Penting untuk diperhatikan adalah logika pemrograman untuk **condition**. **While (\$i <= 10)** juga berarti bahwa jika nilai variabel **\$i = 11**, maka perulangan akan berhenti. Di dalam kode program, kita harus membuat sebuah baris **statement** yang digunakan untuk terus menambahkan nilai **\$i** supaya nilai **\$i** bisa mencapai angka lebih dari 10 untuk menghentikan perulangan.

Setelah membuat beberapa baris kode **echo** untuk menampilkan angka ke web browser pada baris ke-5 dan 6, saya menambahkan kode **\$i=\$i+1** pada baris ke-7. Baris inilah yang akan menambahkan nilai variabel counter **\$i** sebanyak 1 angka pada tiap perulangan, sehingga pada perulangan ke 10, nilai **\$i** akan menjadi 11. Dan kondisi **while** akan menghasilkan **FALSE**, sehingga proses perulangan berhenti.

Kesalahan dalam memahami **logika while** sering menghasilkan perulangan yang akan memproses secara terus menerus (**infinity loop**).

Anda juga bebas menentukan awal dari variabel **counter \$i**, misalnya untuk mulai dari angka 100 dan mundur ke belakang seperti contoh berikut:

```
<?php
$i=100;
while ($i >= 0)
{
echo "$i";
echo "<br />";
$i--=8;
}
?>
```

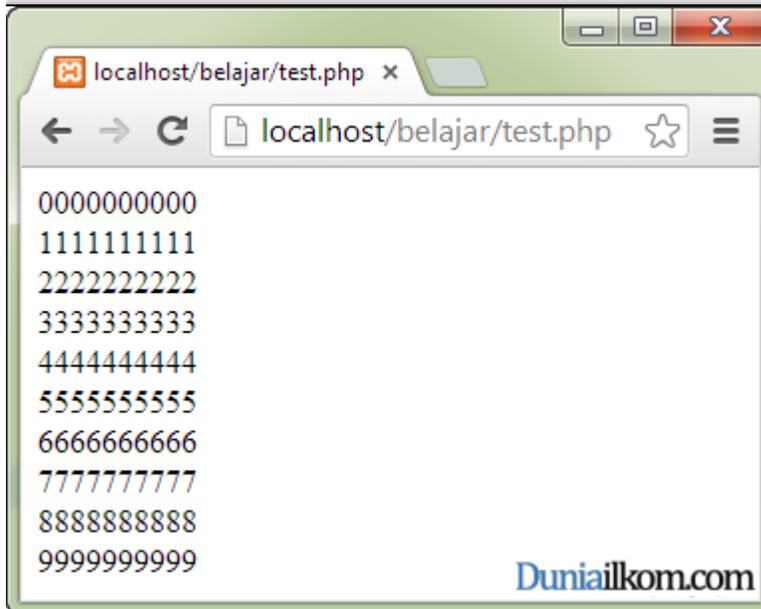


Perulangan while tersebut akan menghasilkan angka menurun dari 100 sampai dengan 0, dimana pada setiap perulangan nilai 100 akan dikurangi dengan 8.

3.6.2 PENULISAN NESTED LOOP UNTUK WHILE

Walaupun **struktur while** agak jarang digunakan untuk **nested loop**, anda bisa membuat perulangan bersarang dengan **struktur while**, seperti contoh berikut ini:

```
<?php
$i=0;
while ($i < 10)
{
    $j=0;
    while ($j < 10)
    {
        echo $i;
        $j++;
    }
    echo "<br />";
    $i++;
}
```



Dalam kode diatas, saya membuat perulangan yang sama seperti contoh **nested loop** pada tutorial **perulangan for**, dimana kode diatas akan menampilkan angka 0-9 sebanyak 9 angka.

Jika anda membandingkan dengan **struktur for** untuk hasil yang sama, perulangan for akan '*lebih*' mudah dipahami dibandingkan perulangan while diatas.

3.6.3 CARA PENULISAN ALTERNATIF STRUKTUR PERULANGAN WHILE

Sebagai cara penulisan alternatif, PHP menyediakan penulisan blok **while** yang biasanya menggunakan tanda *kurung kurawal* dengan **endwhile**.

Berikut adalah contoh penulisan alternatif struktur while dalam PHP:

```
<?php
$i=1;
while ($i <= 10):
echo "$i";
echo "<br />";
$i=$i+1;
endwhile;
?>
```

Untuk penulisan alternatif ini, saya mengganti tanda **kurung kurawal** untuk menandai blok **while** dengan tanda **titik dua** (:) pada awal perulangan, dan perintah **endwhile** di akhir blok.

3.6.4 PENGERTIAN PERULANGAN DO-WHILE DALAM PHP

Perulangan **while** dan **do-while** pada dasarnya hampir sama. Perbedaan terletak pada 'lokasi' pengecekan kondisi perulangan.

Dalam struktur **while**, pengecekan untuk kondisi perulangan di lakukan di awal, sehingga jika kondisi tidak terpenuhi, maka perulangan tidak akan pernah dijalankan.

Namun pada perulangan **do-while**, pengecekan kondisi akan dilakukan di akhir perulangan, sehingga walaupun kondisi adalah **FALSE**, perulangan akan tetap berjalan minimal 1 kali.

Sebagai perbandingan, perhatikan contoh struktur while berikut ini:

```
<?php
$i=1000;
while ($i <= 10)
{
echo "$i";
echo "Tidak akan tampil di browser";
$i=$i+1;
}
?>
```

Kode program diatas tidak akan menampilkan apa-apa, karena kondisi **while (\$i<=10)** sudah menghasilkan nilai **FALSE** pada awal program (karena saya sudah mendefenisikan nilai \$i=1000, pada baris pertama)

Namun jika kode diatas kita ubah menjadi **do-while**, maka berikut hasilnya:

```
<?php
$i=1000;
do
{
echo "$i";
echo "Akan tampil di browser";
```

```
$i=$i+1;
} while ($i <= 10);
?>
```

Program diatas akan menampilkan "1000Akan tampil di browser". Hal ini terjadi karena pada struktur **do-while**, perulangan program akan tampil setidaknya 1 kali walaupun kondisi **while** menghasilkan **FALSE**.

3.6.5 CARA PENULISAN STRUKTUR PERULANGAN DO-WHILE PHP

Penulisan struktur **do-while** mirip dengan **struktur while**, namun kita menambahkan perintah **do** di awal struktur. Berikut adalah format dasar penulisan **struktur do-while** dalam PHP:

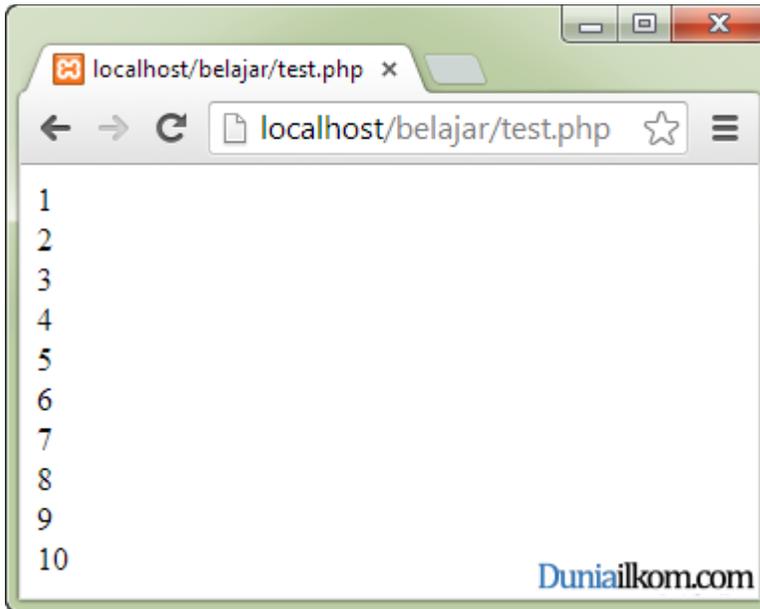
```
do {
    statement;
    statement;
} while (condition);
```

Setelah perintah **do**, di dalam blok kurung kurawal adalah **statement**. **Statement** adalah kode program yang akan diulang. Kita bisa membuat beberapa kode program untuk menampilkan perintah seperti **echo**, atau perintah yang lebih kompleks. Namun di dalam bagian ini harus ada baris program yang digunakan sebagai '*penghenti*' perulangan.

Condition adalah kondisi yang harus dipenuhi agar perulangan berlangsung. Selama **condition** bernilai **TRUE**, maka perulangan akan terus dilakukan. **Condition** ini akan diperiksa pada tiap perulangan, dan hanya jika hasilnya **FALSE**, maka proses perulangan berhenti.

Sebagai contoh sederhana perulangan **do-while**, berikut adalah kode perulangan untuk menampilkan angka 1 sampai dengan 10 pada web browser:

```
<?php
$i=1;
do
{
    echo "$i";
    echo "<br />";
    $i=$i+1;
} while ($i <= 10);
?>
```



Anda juga bisa menggunakan **struktur do-while** untuk perulangan bersarang (**nested loop**) seperti pada tutorial **perulangan for** dan **while**.

Sama seperti perulangan **while**, dalam perancangan [perulangan do-while](#) ini kita harus memahami alur logika program yang dibuat. Karena tidak seperti **perulangan for** dimana jumlah perulangan telah ditentukan di awal, untuk **struktur do-while** banyak perulangan di tentukan pada saat program mencapai kondisi **FALSE**. Kesalahan dalam alur logika akan membuat PHP tidak pernah berhenti memproses perulangan.

3.7 BREAK DALAM PERULANGAN

Ketika proses perulangan berjalan, ada kalanya kita ingin segera keluar dari perulangan jika sebuah kondisi tertentu telah terpenuhi, sehingga sisa proses perulangan tidak perlu dijalankan.

Misalkan kita memiliki nama-nama mahasiswa yang tersimpan di dalam sebuah array atau di dalam database. Proses pencarian sederhana dapat dirancang dengan melakukan pencocokan secara berulang dimulai dari nama pertama, kedua, dan seterusnya. Perulangan ini akan dilakukan sebanyak daftar mahasiswa yang ada.

Akan tetapi, jika nama yang dicari telah ditemukan, proses perulangan seharusnya dapat dihentikan saat itu juga, karena tujuan pencarian nama telah selesai.

Untuk keperluan inilah PHP menyediakan instruksi **break**. **Break** berfungsi sebagai perintah kepada **web server** untuk menghentikan perulangan secara *prematuur*, yaitu menghentikan perulangan di luar dari yang direncanakan.

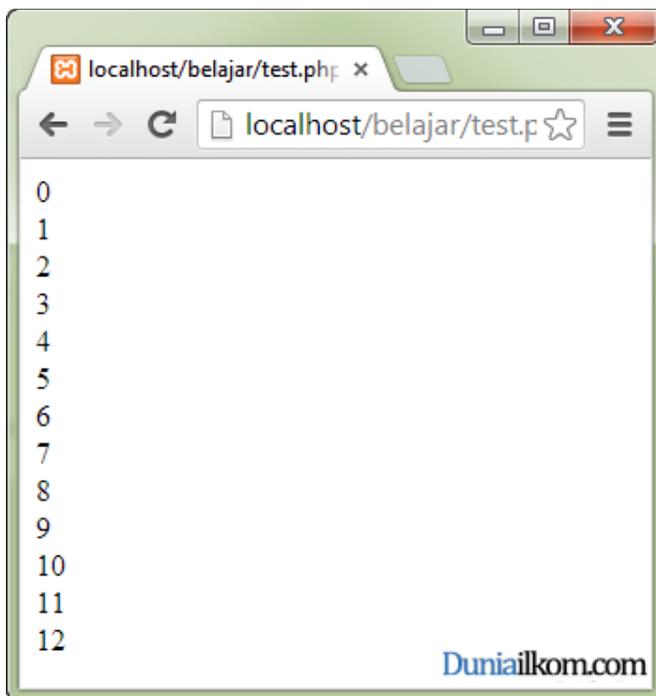
3.7.1 CARA PENULISAN PERINTAH BREAK

Perintah **break** dapat di letakkan di posisi manapun di dalam perulangan, namun biasanya kita akan membuat **logika IF** untuk menentukan kapan perintah **break** akan dijalankan.

3.7.1.1 CONTOH PENGGUNAAN BREAK DALAM PERULANGAN FOR

Berikut adalah contoh program perulangan for dengan menggunakan perintah **break**:

```
<?php
for ($i=0; $i <100; $i++)
{
    if ($i==13)
    {
        break;
    }
    echo $i;
    echo "<br />";
}
```



Dalam program diatas, saya membuat perulangan for dari 0 sampai 100, dan dalam keadaan normal, perintah **for (\$i=0; \$i <100; \$i++)** akan memproses perulangan sebanyak 100 kali. Namun pada baris ke-4 saya menambahkan sebuah **struktur IF** yang menyatakan bahwa jika nilai variabel **counter \$i** sama dengan 13, maka **break**. Perintah **break** akan membuat perulangan **for** langsung dihentikan, dan kita hanya menghasilkan perulangan sampai angka 12. Ini terjadi karena perintah **echo**

\$i ditempatkan setelah **break**. Jika perintah **echo \$i** ini anda pindahkan di baris sebelum **break**, akan tampil angka 13 (yang menandakan kalau perulangan sudah masuk ke **\$i = 13**).

3.7.1.2 CONTOH PENGGUNAAN BREAK DALAM PERULANGAN WHILE

Sama seperti perulangan **for**, perintah **break** digunakan untuk menghentikan perulangan **while** secara *prematur*, atau sebelum kondisi pada bagian **condition** terpenuhi.

Menggunakan contoh yang sama dengan struktur **perulangan for**, berikut adalah contoh penggunaan **break** untuk *perulangan while*:

```
<?php
$i=0;
while ($i < 100)
{
    $i++;
    if ($i==13)
    {
        break;
    }
    echo $i;
    echo "<br />";
}
```

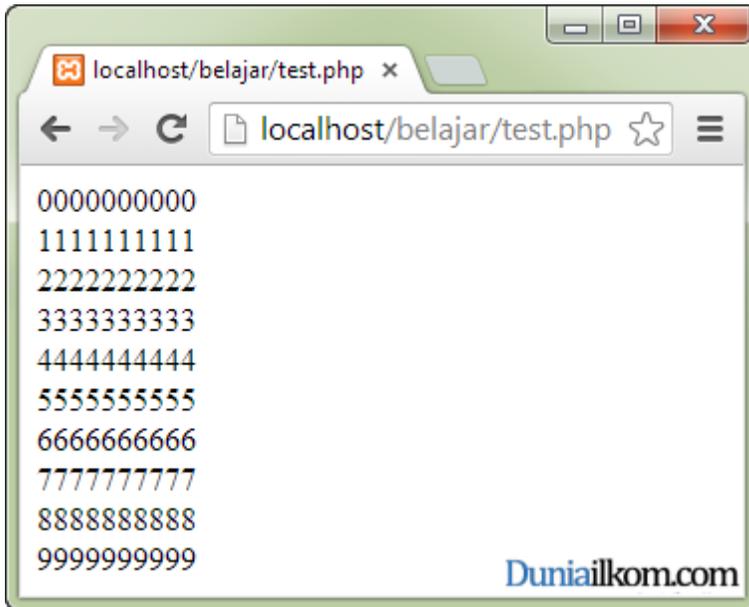
Dalam contoh kode program diatas, perulangan akan berhenti pada iterasi ke-13. Dimana saya membuat sebuah kondisi ketika variabel counter bernilai 13, maka **break**. Perintah **break** akan menghentikan perulangan secara paksa.

3.7.2 CARA PENGGUNAAN INSTRUKSI BREAK PADA PERULANGAN BERSARANG (NESTED LOOP)

Untuk perulangan bersarang (**nested loop**), instruksi **break** bisa bermakna ganda, yaitu apakah kita ingin menghentikan perulangan luar, atau perulangan dalam.

Agar lebih mudah dipahami, berikut adalah contoh **nested loop** yang pernah kita bahas pada tutorial perulangan **for**:

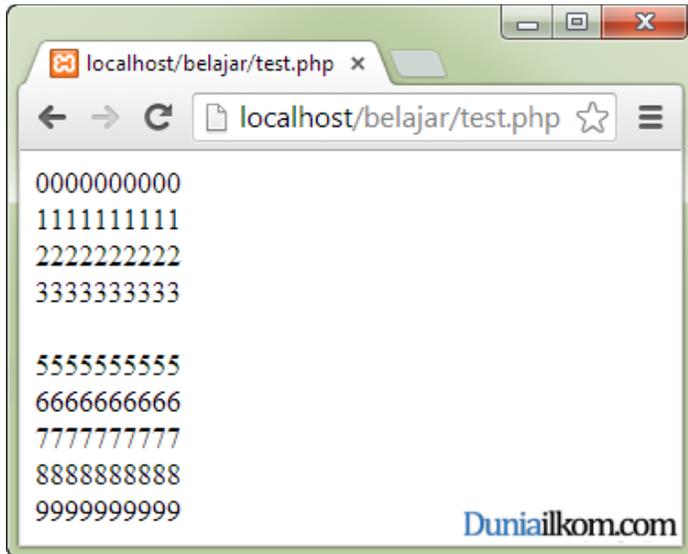
```
<?php
for ($i=0; $i <10; $i++)
{
    for ($j=0; $j <10; $j++)
    {
        echo $i;
    }
    echo "<br />";
}
```



Dalam perulangan tersebut, variabel **counter \$i** digunakan untuk perulangan luar (**outer loop**), dan variabel **counter \$j** digunakan untuk perulangan dalam (**inner loop**).

Jika kita membuat perintah **break** pada perulangan **\$j (inner loop)**, maka yang akan dihentikan hanya perulangan **\$j** saja, seperti pada contoh program berikut ini:

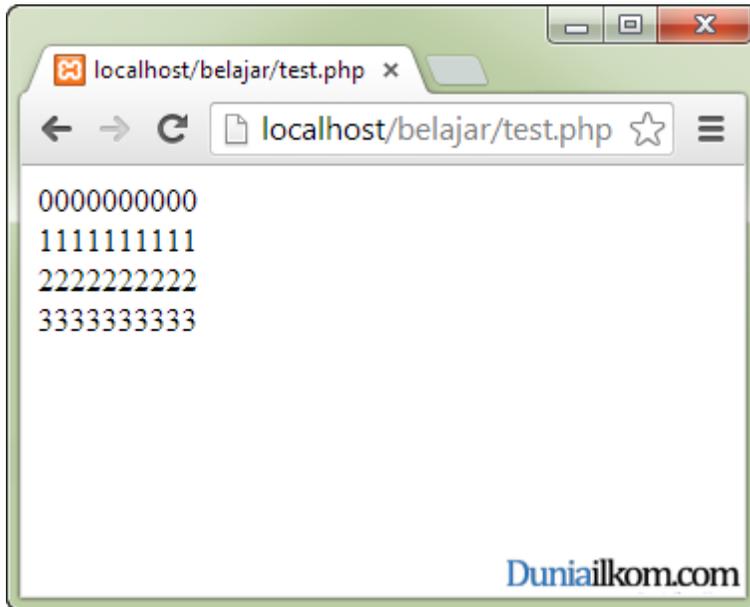
```
<?php
for ($i=0; $i <10; $i++)
{
    for ($j=0; $j <10; $j++)
    {
        if ($i==4)
        {
            break;
        }
        echo $i;
    }
    echo "<br />";
}
```



Dengan memberikan perintah **break** pada perulangan **\$j**, maka perulangan **\$j** akan berhenti pada angka 4, namun perulangan **\$i** akan terus berjalan. Bagaimana jika kita juga ingin menghentikan perulangan **\$i**? Caranya adalah dengan mengubah perintah **break** diatas menjadi **break 2**. Angka 2 bertujuan untuk memberitahu PHP bahwa perintah break ditujukan untuk 2 level perulangan diatasnya.

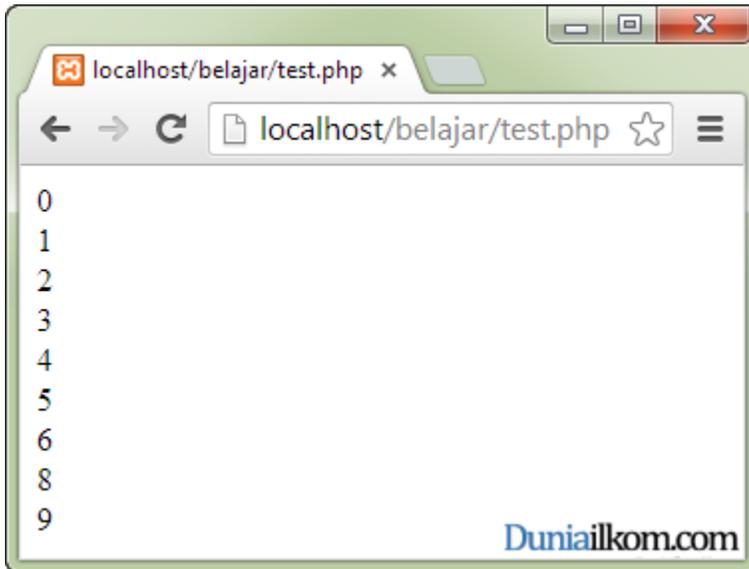
Berikut adalah contoh program sebelumnya, dengan ditambahkan break 2:

```
<?php
for ($i=0; $i <10; $i++)
{
    for ($j=0; $j <10; $j++)
    {
        if ($i==4)
        {
            break 2;
        }
        echo $i;
    }
}
echo "<br />";
}
```



Jika anda membuat **nested loop** dengan 3 tingkatan, atau 3 level, maka kita bisa menggunakan perintah **break 3** untuk keluar dari perulangan terdalam. Perintah **continue** juga digunakan untuk menginterupsi perulangan dalam PHP, namun jika perintah **break** digunakan untuk menghentikan perulangan, maka perintah **continue** hanya akan menghentikan perulangan untuk 1 iterasi saja, lalu proses perulangan akan dilanjutkan. Berikut contoh kode PHP penggunaan perintah **continue**:

```
<?php
for ($i=0; $i <10; $i++)
{
    if ($i==7)
    {
        continue;
    }
    echo $i;
    echo "<br />";
}
```



Contoh perulangan diatas mirip dengan contoh pada perulangan break. Setelah perintah **for**, saya membuat sebuah kondisi **IF** yang jika **variabel counter \$i** bernilai 7, maka jalankan **continue**.

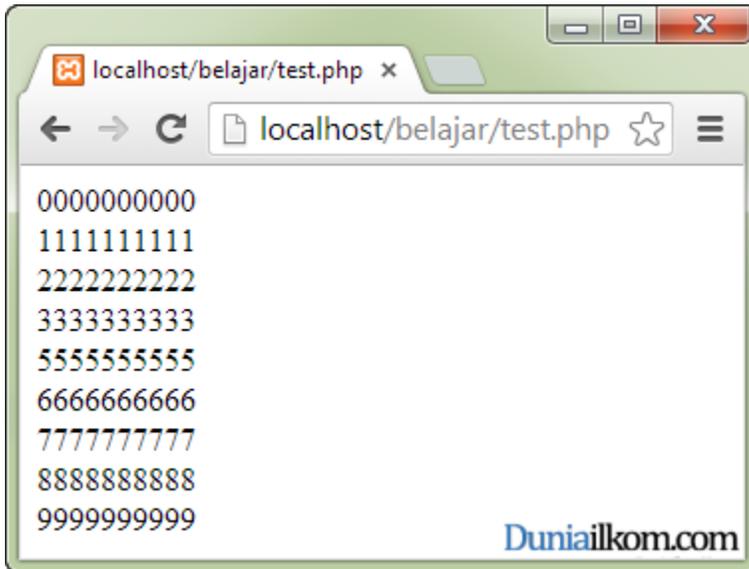
Arti dari **continue** ini adalah sebuah instruksi kepada PHP untuk melewati sisa perintah dalam perulangan, dan langsung lompat ke nilai counter berikutnya, yakni 8. Dari hasil program, anda tidak akan melihat angka 7 ditampilkan.

3.7.3 CARA PENGGUNAAN CONTINUE PADA NESTED LOOP

Sama seperti perintah **break**, perintah **continue** juga bisa digunakan untuk **nested loop**, dan kita menggunakan angka setelah perintah **continue** untuk menginstruksikan **level kedalaman loop**.

Berikut adalah contoh penggunaan perintah continue dalam nested loop

```
<?php
for ($i=0; $i <10; $i++)
{
    for ($j=0; $j <10; $j++)
    {
        if ($i==4)
        {
            continue 2;
        }
        echo $i;
    }
}
echo "<br />";
}
```



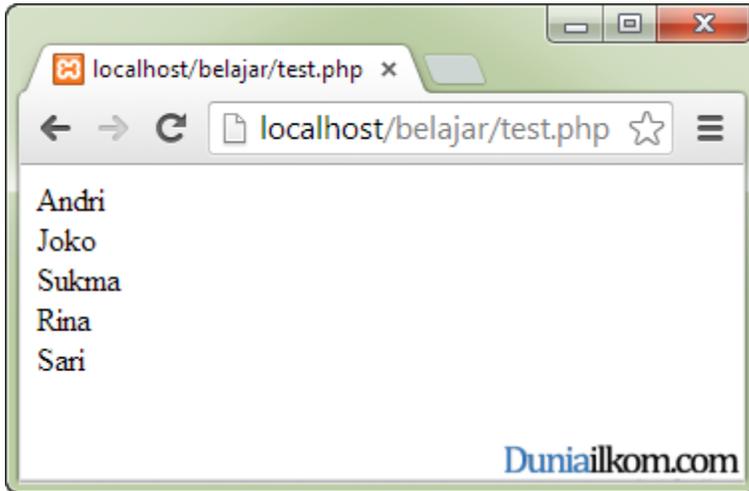
Dalam contoh diatas, perintah **continue 2** akan menginstruksikan kepada PHP untuk tidak mengeksekusi perulangan for pada perulangan terluar, yakni jika **\$i** sama dengan 4, dan melanjutkan kedalam iterasi selanjutnya, yakni **\$i=5**.

3.8 FOREACH

Array merupakan tipe data yang sering digunakan dalam membuat program menggunakan PHP. Kemampuan **array** dalam menyimpan banyak data dalam satu variabel akan sangat berguna untuk menyederhanakan dan menghemat penggunaan variabel.

Untuk menampilkan dan memproses data dari **array**, kita bisa memanfaatkan perulangan for, seperti contoh berikut ini:

```
<?php
$nama = array("Andri","Joko","Sukma","Rina","Sari");
for ($i=0; $i <5; $i++)
{
echo "$nama[$i]";
echo "<br />";
}
?>
```



Contoh diatas membuat perulangan for sebanyak 5 kali, dengan variabel counter **\$i** dimulai dari angka 0 (karena index **array** dimulai dari angka 0). Namun sebagai cara alternatif untuk menampilkan **array**, saya akan mengubah kode diatas dengan menggunakan perulangan **foreach**:

```
<?php
$nama = array("Andri","Joko","Sukma","Rina","Sari");
foreach ($nama as $val)
{
echo "$val";
echo "<br />";
}
?>
```

Perulangan **foreach** diatas akan menampilkan semua isi array dengan perintah yang lebih singkat daripada menggunakan **perulangan for**.

3.8.1 CARA PENULISAN PERULANGAN FOREACH DALAM PHP

Perulangan foreach merupakan perulangan khusus untuk pembacaan nilai **array**. Seperti yang telah kita bahas pada tutorial tentang tipe data array: Mengenal Tipe Data Array dan Cara Penulisan Array dalam PHP, setiap array memiliki pasangan **key** dan **value**. **Key** adalah '*posisi*' dari **array**, dan **value** adalah '*isi*' dari **array**. Format dasar perulangan **foreach** adalah:

```
foreach ($nama_array as $value)
{
    statement (...$value...)
}
```

\$nama_array adalah nama dari **array** yang telah didefinisikan sebelumnya. **\$value** adalah nama '*variabel perantara*' yang berisi data array pada perulangan tersebut. Anda bebas memberikan nama untuk variabel perantara ini, walaupun pada umumnya banyak programmer menggunakan **\$value**, atau **\$val** saja. Berikut adalah contoh perulangan **foreach** sebelumnya:

```

<?php
$nama = array("Andri","Joko","Sukma","Rina","Sari");
foreach ($nama as $val)
{
echo "$value";
    echo "<br />";
}
?>

```

Pada contoh diatas, saya mendefinisikan variabel array **\$nama** dengan format singkat, dan tanpa mendefinisikan **key** secara tertulis. Variabel **\$val** merupakan *variabel perantara* dalam contoh diatas. Perulangan tersebut akan diulang sebanyak data yang terdapat di dalam array, sehingga kita tidak perlu harus menghitung seberapa banyak perulangan yang harus dilakukan.

Jika anda membutuhkan nilai **key** dari **array** untuk dapat diproses, maka PHP menyediakan bentuk kedua dari perulangan **foreach**, dengan format dasar penulisan sebagai berikut:

```

foreach ($nama_array as $key => $value)
{
    statement ($key...$value...)
}

```

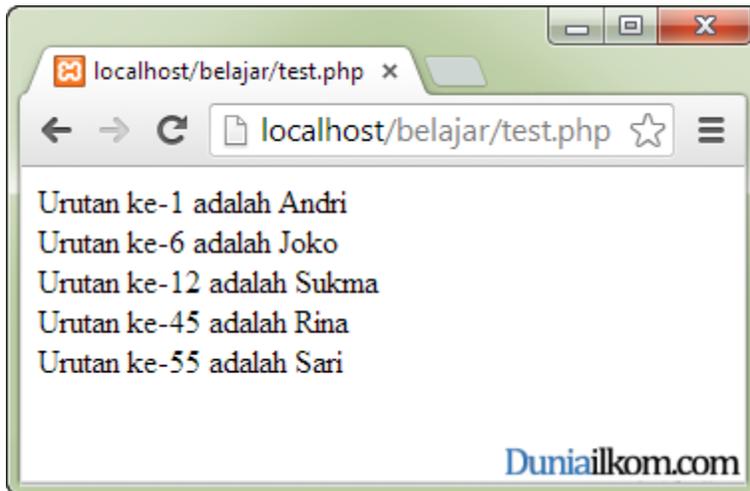
Perbedaan dengan format sebelumnya, disini PHP menyediakan variabel perantara kedua, yaitu variabel **\$key**. Variabel **\$key** ini menampung nilai **key** dari array.

Berikut adalah contoh penggunaannya:

```

<?php
$nama = array(
    1=>"Andri",
    6=>"Joko",
    12=>"Sukma",
    45=>"Rina",
    55=>"Sari");
foreach ($nama as $kunci =>$isi)
{
echo "Urutan ke-$kunci adalah $isi";
echo "<br />";
}
?>

```



Variabel array **\$nama** saya defenisikan menggunakan key yang berbeda-beda. Pada perulangan **foreach**, saya membuat variabel perantara **\$kunci =>\$isi**, sehingga didalam perulangan, variabel **\$kunci** akan berisi **key** dari array, dan variabel **\$isi** akan berisi **nilai** dari array.

Proses menampilkan dan memproses **array** akan lebih mudah dengan menggunakan [perulangan foreach](#) dibandingkan perulangan dasar seperti **for**. Terlebih lagi kita tidak perlu mencari tau seberapa banyak perulangan harus dilakukan, karena perulangan **foreach** akan otomatis berhenti pada data terakhir dari array.

4.1 PENGERTIAN

Dalam merancang kode program, kadang kita sering membuat kode yang melakukan tugas yang sama secara berulang-ulang, seperti membaca tabel dari database, menampilkan penjumlahan, dan lain-lain. Tugas yang sama ini akan lebih efektif jika dipisahkan dari program utama, dan dirancang menjadi sebuah **fungsi**.

Fungsi (atau *Function*) dalam bahasa pemrograman adalah *kode program yang dirancang untuk menyelesaikan sebuah tugas tertentu, dan merupakan bagian dari program utama*. Kita dapat membuat **fungsi** sendiri, atau menggunakan **fungsi** yang dibuat oleh programmer lain.

Dalam dunia pemrograman terdapat istilah '*lazy programming*' yang artinya bukanlah programmer yang malas. Tetapi, daripada membuat kode program umum dari dasar, kita bisa menggunakan **fungsi** yang telah dibuat oleh programmer lain. PHP bahkan menyediakan ribuan fungsi bawaan yang tersedia untuk membantu kita dalam merancang program.

Mengetahui cara penggunaan fungsi ini akan menghemat waktu pembuatan program dan membuat kode program menjadi lebih efisien. *Lazy programming is smart programming*.

4.1.1 CARA MENGGUNAKAN FUNGSI PHP

Menggunakan fungsi dalam teori pemrograman sering juga disebut dengan istilah '*memanggil fungsi*' (**calling a function**). **Fungsi** dipanggil dengan menulis nama dari fungsi tersebut, dan diikuti dengan **argumen** (jika ada). **Argumen** ditulis di dalam tanda kurung, dan jika jumlah **argumen** lebih dari satu, maka diantaranya dipisahkan oleh karakter koma.

Setelah memproses nilai inputan, hampir semua fungsi akan memberikan nilai hasil pemrosesan tersebut (walaupun ada fungsi yang tidak memberikan nilai). Cara fungsi memberikan nilainya ini sering disebut dengan '*mengembalikan nilai*' (**return a value**). Nilai yang *dikembalikan* oleh sebuah fungsi dapat ditampung ke dalam variabel, atau langsung ditampilkan ke web browser.

4.1.2 PENGERTIAN ARGUMEN DAN PARAMETER DALAM FUNGSI PHP

Sebuah **fungsi** dalam memproses data, kadang memerlukan beberapa inputan atau nilai masukan. Inputan inilah yang dimaksud dengan **argumen**. Sebuah fungsi bisa membutuhkan 1, 2, atau 5 **argumen**, namun bisa juga tidak memerlukan **argumen** sama sekali.

Parameter adalah sebutan lain untuk **argumen**. Perbedaannya, **parameter** merujuk kepada inputan fungsi pada saat pendefinisian *fungsi* tersebut, dan **argumen** adalah sebutan untuk inputan fungsi pada saat *pemanggilan* fungsi. Kita akan membahas perbedaan **Argumen** dan **Parameter** secara lebih dalam pada tutorial selanjutnya, namun pada dasarnya **argumen** dan **parameter** merujuk kepada hal yang sama, yaitu inputan kepada fungsi dan kedua istilah ini sering dipertukarkan.

4.1.3 CONTOH PEMANGGILAN FUNGSI PHP

Sebagai latihan dan praktker dalam menggunakan fungsi, Berikut adalah format dasar pemanggilan, dan pengembalian nilai fungsi:

```
$varibel_hasil_fungsi = nama_fungsi(argumen1, argumen2, argumen3)
```

- **\$varibel_hasil_fungsi** adalah variabel yang akan menampung hasil pemrosesan fungsi. Tergantung fungsinya, hasil dari sebuah fungsi bisa berupa angka, string, array, bahkan objek.
- **nama_fungsi** adalah nama dari fungsi yang akan dipanggil
- **argumen1, argumen2** adalah nilai inputan fungsi. Banyaknya **argumen** yang dibutuhkan, tergantung kepada fungsi tersebut. Jika sebuah fungsi membutuhkan **argumen** 2 buah angka, maka kita harus menginputnya sesuai dengan aturan tersebut, atau jika tidak, PHP akan mengeluarkan error.

Sebagai contoh, PHP menyediakan *fungsi akar kuadrat*, yakni **sqrt()**, berikut adalah cara penggunaannya:

```
<?php
$akar_kuadrat = sqrt(49);
echo "Akar kuadrat dari 49 adalah $akar_kuadrat";
// Akar kuadrat dari 49 adalah 7
?>
```

Dalam contoh diatas, fungsi **sqrt()** akan menghitung akar kuadrat dari nilai argumen yang diinput. Saya menambahkan argumen 49 sebagai inputan. Nilai hasil dari **fungsi sqrt(49)**, selanjutnya di tampung dalam variabel **\$akar_kuadrat**, yang kemudian ditampilkan ke dalam web browser. Selain ditampung di dalam variabel, kita bisa menampilkan hasil fungsi langsung ke web browser, seperti contoh berikut:

```
<?php
echo "12 pangkat 2 adalah: ".pow(12,2);
// 12 pangkat 2 adalah: 144
?>
```

Fungsi **pow()** adalah fungsi pemangkatan matematika bawaan PHP. Fungsi ini membutuhkan 2 argumen, argumen pertama adalah nilai awal yang ingin dihitung, dan **argumen** kedua adalah nilai pangkat. **Pow(12,2)** sama dengan 12 kuadrat.

Perlu juga diperhatikan adalah **tipe parameter** yang dibutuhkan oleh sebuah fungsi. Seperti 2 contoh kita diatas, fungsi **sqrt()** dan **pow()** adalah fungsi *matematika*. Kedua fungsi ini hanya bisa memproses parameter dengan tipe angka (**integer** dan **float**). Jika anda memasukkan parameter jenis **string**, maka PHP akan mengeluarkan error.

Jumlah dan urutan **argumen** juga harus sesuai dengan yang dibutuhkan oleh fungsi. Jika sebuah fungsi hanya membutuhkan 1 argumen, maka kita tidak bisa menambahkan argumen kedua, kecuali ada argumen yang bersifat OPSIONAL (dapat diabaikan).

4.2 CARA PENULISAN FUNGSI DAN PEMBUATANNYA

Sebuah fungsi merupakan kode program yang dirancang untuk menyelesaikan sebuah tugas tertentu. Tujuan memisahkan sebuah kode menjadi **fungsi** adalah untuk kepraktisan dan kemudahan dalam membuat **program utama**. Karena jika dijadikan **fungsi**, maka untuk menjalankan tugas yang sama, kita tinggal memanggil fungsi tersebut, tanpa perlu membuat kembali kode programnya.

Untuk **membuat fungsi** di dalam PHP, berikut adalah format dasar pembuatan fungsi:

```
function nama_fungsi ($parameter1, $parameter2)
{
// kode program fungsi
return $nilai_akhir
}
```

- Kata **function** adalah *instruksi* kepada PHP bahwa kita akan membuat **fungsi**
- **nama_fungsi** adalah nama dari fungsi yang akan ditulis
- **\$parameter1, \$parameter2** adalah variabel perantara yang akan menyimpan inputan yang diperlukan dalam pemrosesan fungsi (argumen). Tergantung kebutuhan, anda bebas merancang seberapa banyak **parameter** yang dibutuhkan.
- **return** adalah perintah khusus untuk fungsi, dimana kata **return** menginstruksikan kepada PHP bahwa pemrosesan fungsi telah selesai. **return \$nilai_akhir** berarti bahwa fungsi akan *'mengembalikan'* **\$nilai_akhir** sebagai hasil dari fungsi.

Perhatikan juga bahwa fungsi ini berada di dalam blok program yang ditandai dengan kurung kurawal pada baris pertama dan terakhir fungsi. Agar lebih mudah dipahami, kita akan mencoba membuat beberapa fungsi PHP sederhana.

4.2.1 CONTOH PEMBUATAN FUNGSI PHP

Sebagai contoh pertama **fungsi PHP**, saya akan membuat fungsi sederhana untuk perkalian 2 angka. Nama fungsi ini adalah **perkalian**, dan membutuhkan 2 argumen dengan tipe data angka (**integer** atau **float**), berikut adalah cara pembuatan dan penggunaan fungsi perkalian:

```
<?php
//pembuatan fungsi
function perkalian($angka1, $angka2)
{
$a= $angka1;
$b= $angka2;
$hasil= $a*$b;
return $hasil;
}
//pemanggilan fungsi
$hasil=perkalian(4,5);
echo "Perkalian 4 x 5 adalah $hasil";
echo "<br />";
echo "Perkalian 7 x 2 adalah ".perkalian(7,2);
?>
```

Pada baris ke-3, saya mendefinisikan fungsi **perkalian()** yang memerlukan 2 parameter. **\$Parameter** ditulis sebagai variabel, dan anda bebas menentukan nama variabel ini, dalam contoh diatas, parameter untuk fungsi **perkalian()** adalah **\$angka1** dan **\$angka2**.

Di dalam fungsi **perkalian()**, saya membuat variabel **\$a** dan **\$b** yang digunakan untuk menampung nilai **\$angka1** dan **\$angka2**, pemindahan variabel ini sebenarnya tidak diperlukan, namun anda akan sering menemukan hal ini di dalam berbagai fungsi. Biasanya pemindahan ini dilakukan agar lebih mudah dan singkat dalam menggunakan variabel pada pemrosesan fungsi.

Variabel **\$hasil** digunakan untuk menampung nilai akhir dari perkalian **\$a*\$b**, dan hasilnya dikembalikan dengan perintah **return \$hasil**. **Return** secara otomatis menutup fungsi, dan jika anda masih memiliki kode program setelah perintah **return**, perintah tersebut tidak akan diproses, oleh karena itu perintah **return** harus diletakkan di akhir penulisan fungsi.

Selanjutnya pada baris ke-12, saya memanggil fungsi **perkalian()** dengan menyimpan nilai kembaliannya ke dalam variabel **\$hasil**, atau bisa dipanggil secara langsung dalam satu baris perintah **echo**, seperti pada baris ke-15.

Sebagai contoh fungsi kedua, saya akan membuat fungsi untuk menentukan luas lingkaran. Seperti yang kita ketahui, luas lingkaran didapat dengan rumus: **pi*jari-jari*jari-jari**. Berikut adalah program untuk menghitung luas lingkaran:

```
<?php
//pemanggilan fungsi
echo "Luas Lingkaran dengan jari-jari 7cm = ".luas_lingkaran(7)."cm";
//pembuatan fungsi
function luas_lingkaran($jari2)
{
return M_PI*$jari2*$jari2;
}
?>
```

Saya memulai program dengan langsung memanggil fungsi **luas_lingkaran(7)**, padahal fungsi tersebut belum didefinisikan. Hal ini dimungkinkan karena pada saat kode PHP dijalankan, web server akan **compile** fungsi-fungsi yang ada terlebih dahulu. Sehingga kita bisa memanggil fungsi yang di definisikan setelah dipanggil.

Pada fungsi **luas_lingkaran()**, satu-satunya **argumen** yang diperlukan adalah panjang jari-jari lingkaran. **M_PI** adalah konstanta matematis yang disediakan PHP, yang nilainya sama dengan konstanta pi, yaitu 3,14. Namun anda juga bisa mengubahnya menjadi $3.14 * \$jari2 * \$jari2$.

4.2.2 PERBEDAAN ANTARA ARGUMEN DAN PARAMETER

Pada penjelasan tentang fungsi **perkalian()** dan **luas_lingkaran()**, saya menggunakan 2 istilah untuk nilai inputan fungsi, yakni **argumen** dan **parameter**. Tetapi apa perbedaan keduanya?

Argumen merujuk kepada inputan fungsi pada saat fungsi dipanggil, contohnya pada saat saya memanggil fungsi **perkalian(4,5)**, angka **4** dan **5** disebut **argumen**.

Sedangkan **Parameter** merujuk kepada inputan fungsi pada saat pendefinisian. Saya membuat fungsi perkalian dengan perintah : **function perkalian(\$angka1, \$angka2)**, variabel **\$angka1** dan **\$angka2** adalah **parameter**.

Namun pada penggunaan sehari-hari, istilah **parameter** dan **argumen** sering dipertukarkan. Termasuk dalam manual resmi PHP, dimana istilah argumenlah yang sering digunakan. Dalam tutorial di duniaikom, saya akan menggunakan istilah PARAMETER dan ARGUMEN secara bergantian.

4.3 VARIABEL SCOPE, GLOBAL, DAN STATIC

Variabel Scope (atau ruang lingkup variabel) adalah jangkauan kode program dimana perintah program masih bisa mengakses sebuah variabel.

Jika kita mendefinisikan sebuah variabel pada satu file PHP, maka variabel tersebut dapat diakses oleh seluruh kode program pada halaman yang sama. Namun jika variabel tersebut di

definisikan di dalam sebuah fungsi, variabel itu belum tentu bisa diakses dari luar fungsi tersebut. Hal inilah yang dimaksud dengan **Variabel Scope**.

Variabel yang didefinisikan di dalam sebuah fungsi, secara **default tidak dapat diakses** oleh kode program di luar fungsi tersebut. Dan begitu juga sebaliknya, variabel yang didefinisikan di luar fungsi, **tidak bisa diakses** dari dalam fungsi.

4.3.1 CONTOH VARIABEL SCOPE DALAM FUNGSI PHP

Untuk memahami konsep variabel scope, berikut adalah contoh kode program dalam PHP:

```
<?php
$a = 5;

function coba()
{
    $a=10;
    $b=7;
}

// pemanggilan fungsi coba()
coba();
echo $a; // 5
echo $b; // error:notice
?>
```

Pada baris ke-2, saya mendefinisikan variabel **\$a**, dan memberikan nilai awal = 5. Pada baris ke-4 saya membuat fungsi **coba()** dan mendefinisikan kembali variabel **\$a** yang kali ini nilainya adalah 10, dan juga membuat sebuah variabel baru, yakni **\$b**.

Setelah memanggil fungsi **coba()** pada baris ke-11, saya kemudian memeriksa nilai **\$a** dengan perintah **echo**. Dan ternyata nilai **\$a** adalah 5, bukan 10. Dan ketika saya ingin mengakses variabel **\$b**, PHP akan mengeluarkan peringatan bahwa variabel **\$b** belum di defenisikan, dengan pesan error: NOTICE: UNDEFINED VARIABLE: B IN D:\XAMPP\HTDOCS\BELAJAR\TEST.PHP ON LINE 13.

Hal ini terjadi karena variabel **\$a** dan **\$b** berada di dalam fungsi **coba()** yang merupakan variabel yang berbeda dengan variabel **\$a** yang berada diluar fungsi. Jangkauan variabel **\$a** dan **\$b** hanya berada di dalam fungsi.

Contoh lainnya, perhatikan kode program berikut ini:

```
<?php
$b = 7;
```

```
function coba()
{
    $a=10;
    echo $a;
    echo $b;
}

coba();
?>
```

Pada program kali ini, saya mencoba mengakses variabel **\$b** dari dalam fungsi **coba()**, namun PHP akan mengeluarkan pesan peringatan pada baris ke-8: *Notice: Undefined variable: b in D:\xampp\htdocs\belajar\test.php on line 8* yang berarti bahwa PHP tidak menemukan adanya variabel **\$b**. Variabel **\$b** hanya dapat diakses dalam ruang lingkup di luar fungsi **coba()**.

Konsep pembatasan **variabel scope** ini terkesan merepotkan, namun sebenarnya sangat berguna untuk men-*isolasi* penggunaan variabel agar tidak saling '*menimpa*'. Fungsi-fungsi bawaan di dalam PHP dibuat oleh berbagai programmer dari seluruh dunia, dan mungkin saja kita secara tidak sengaja menggunakan nama variabel yang sama dengan nama variabel yang ada dalam salah satu fungsi tersebut. Dengan menerapkan **variabel scope**, PHP dapat terhindar dari permasalahan tersebut.

4.3.2 PENGERTIAN GLOBAL VARIABEL

Jika kita tetap ingin menggunakan variabel yang didefinisikan di luar fungsi dan sebaliknya, PHP memperbolehkan hal tersebut dengan menambahkan sebuah kata kunci '**global**' sebelum pendefinisian nama variabel.

Berikut contoh penggunaan keyword global:

```
<?php
$a = 7;

function coba()
{
    global $a;
    global $b;
    $b=15;
    echo $a; //7
    echo $b; //15
}

coba();
echo $a; //7
echo $b; //15
?>
```

Program diatas tidak akan menghasilkan error seperti sebelumnya, dan kita bisa mengakses nilai variabel **\$a** dari dalam fungsi **coba()**, dan nilai variabel **\$b** di luar fungsi.

Kata kunci global membuat fungsi dapat mengakses variabel yang didefinisikan diluar fungsi.

4.3.3 PENGERTIAN STATIC VARIABEL

Static Variabel, atau *variabel statis* adalah jenis variabel yang mempertahankan nilainya pada setiap pemanggilan fungsi. Untuk variabel normal, nilai dari variabel tersebut akan secara otomatis dihapus pada saat fungsi selesai dijalankan, dan akan dibuat ulang pada saat fungsi dipanggil.

Namun jika sebuah variabel dinyatakan sebagai **static variabel**, maka nilai variabel tersebut akan tetap dipertahankan walaupun fungsi telah selesai dijalankan. Biasanya fungsi ini dimanfaatkan jika kita ingin menghitung berapa kali sebuah fungsi dipanggil.

Berikut adalah contoh program penggunaan variabel statis dalam PHP:

```
<?php
function coba()
{
    static $a=0;
    $a=$a+1;
    return "Ini adalah pemanggilan ke- $\$a$  fungsi coba() <br />";
}

echo coba();
echo coba();
echo coba();
echo coba();
?>
```

Jika anda menghapus **keyword static** pada baris ke-4, variabel **\$a** akan selalu bernilai 1, karena terdapat operasi $\$a=0$ dan $\$a=\$a+1$ pada setiap kali pemanggilan fungsi **coba()**. Namun dengan membuat **\$a** sebagai **static variable**, nilai dari **\$a** akan terus dipertahankan sepanjang pemrosesan halaman oleh PHP.

Konsep tentang variabel scope, global variabel dan static variabel ini akan membantu kita dalam memahami cara kerja fungsi dalam PHP.

4.4 PENGECEKAN TIPE DATA ARGUMEN

4.4.1 PENTINGNYA PENGECEKAN TIPE DATA ARGUMEN

Dalam pembuatan **fungsi PHP**, selain merancang cara kerja fungsi, kita juga harus memperkirakan berapa banyak **parameter** yang dibutuhkan untuk fungsi tersebut. Sebuah fungsi bisa memiliki 1, 2 atau 5 **parameter**, namun bisa juga tanpa **parameter** sama sekali.

Tergantung tujuannya, sebuah fungsi umumnya hanya memperbolehkan tipe data tertentu sebagai **argumen**. Misalnya, untuk fungsi yang berhubungan dengan matematika, biasanya hanya membutuhkan argumen dengan tipe data angka (**integer** atau **float**), dan fungsi penghitung kata, hanya membutuhkan tipe data **string** sebagai argumen.

Jika anda adalah satu-satunya pengguna fungsi yang anda rancang sendiri, maka dapat dipastikan bahwa tidak akan ada inputan argumen yang salah tipe data. Namun jika terdapat kemungkinan fungsi yang dirancang akan digunakan oleh pihak lain, pengecekan tipe data argumen perlu dirancang agar fungsi berjalan sebagaimana mestinya.

Jika tipe data **parameter** tidak sesuai, maka fungsi tidak akan berjalan sebagaimana mestinya, dan biasanya PHP akan mengeluarkan pesan *error*. Cara elegan untuk mengatasi permasalahan ini adalah membuat kode program untuk memeriksa tipe data **parameter** ini sebelum masuk kepada pemrosesan di dalam fungsi.

Pengecekan tipe data dilakukan pada awal pemrosesan **fungsi**, dan jika tipe data tidak sesuai, kita bisa membuat pesan bahwa fungsi tidak dapat diproses. Pengecekan apakah suatu **argumen** merupakan bagian dari tipe data tertentu, dilakukan dengan fungsi khusus yang telah disediakan PHP.

Berikut adalah list fungsi pengecekan tipe data dalam PHP:

- **is_array(\$var)**: fungsi pengecekan apakah tipe data adalah array
- **is_bool(\$var)**: fungsi pengecekan apakah tipe data adalah boolean
- **is_double(\$var)**: fungsi pengecekan apakah tipe data adalah float
- **is_float(\$var)**: fungsi pengecekan apakah tipe data adalah float
- **is_int(\$var)**: fungsi pengecekan apakah tipe data adalah integer
- **is_integer(\$var)**: fungsi pengecekan apakah tipe data adalah integer
- **is_long(\$var)**: fungsi pengecekan apakah tipe data adalah integer
- **is_null(\$var)**: fungsi pengecekan apakah tipe data adalah null
- **is_numeric(\$var)**: fungsi pengecekan apakah tipe data adalah angka (integer dan float)
- **is_object(\$var)**: fungsi pengecekan apakah tipe data adalah objek
- **is_real(\$var)**: fungsi pengecekan apakah tipe data adalah float
- **is_resource(\$var)**: fungsi pengecekan apakah tipe data adalah **resource** (seperti variabel yang menampung koneksi ke database)

- **is_scalar(\$var)**: fungsi pengecekan apakah tipe data adalah **scalar** (scalar adalah penyebutan untuk tipe data dasar, seperti integer, float, string atau boolean. Array, object dan resource bukan scalar)
- **is_string(\$var)**: fungsi pengecekan apakah tipe data adalah string

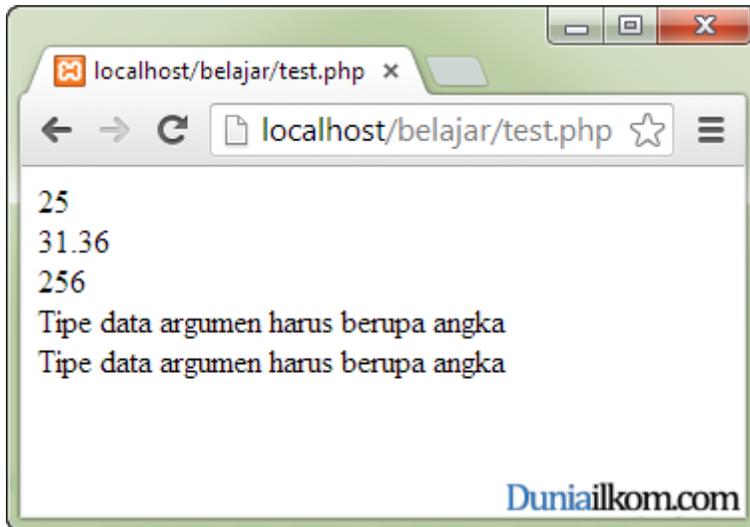
4.4.2 CARA PENGECEKAN TIPE DATA ARGUMEN FUNGSI

Fungsi-fungsi diatas dapat dimanfaatkan untuk pengecekan tipe data suatu variabel, dan tentu saja juga argumen fungsi. Agar lebih mudah dipahami, saya telah merancang fungsi **pangkat()** yang berfungsi untuk melakukan pemangkatan bilangan.

Fungsi **pangkat()** saya rancang dengan 2 buah inputan atau **parameter**. **Parameter** pertama adalah angka yang akan dihitung, dan **parameter** kedua adalah nilai pangkatnya. **pangkat(2,3)** berarti 2 pangkat 3. **pangkat(2,8)** berarti 2 pangkat 8. Kedua parameter ini harus berupa angka, dan khusus untuk nilai pangkat, harus berupa angka bulat (*integer*).

Berikut adalah kode program fungsi pangkat():

```
<?php
function pangkat($nilai, $pangkat)
{
if (is_numeric($nilai) AND is_int($pangkat)) //pengecekan tipe data argumen
{
//Jika argumen sesuai, maka jalankan proses fungsi
$hasil=1;
for ($i=1;$i<=$pangkat;$i++)
{
    $hasil=$hasil*$nilai;
}
return $hasil;
}
else
{
    //Bagian ini akan dijalankan jika tipe data argumen bukan angka
return "Tipe data argumen harus berupa angka";
}
}
//Test beberapa kasus inputan untuk fungsi pangkat()
echo pangkat(5,2);
echo "<br />";
echo pangkat(5.6,2);
echo "<br />";
echo pangkat(2,8);
echo "<br />";
echo pangkat(5,2.9);
echo "<br />";
echo pangkat("lima",2);
echo "<br />";
?>
```



Fungsi **pangkat()** diatas terasa sedikit panjang, namun jika anda telah mengikuti seluruh tutorial PHP di duniaikom, maka fungsi tersebut tidak akan terlalu sulit untuk dipahami. Fungsi **pangkat()** saya rancang untuk menghitung pangkat dari sebuah angka. Variabel **\$nilai** dan **\$pangkat** adalah **parameter** yang akan menjadi *variabel perantara*.

Pada baris ke-4 saya membuat pengecekan masing-masing parameter di dalam logika IF. Fungsi **is_numeric()** dan **is_int()** akan menghasilkan nilai TRUE jika keduanya benar, sehingga saya menggabungkan keduanya kedalam logika **AND**. Seandainya logika **AND** ini salah, maka kondisi IF akan bernilai **FALSE**, dan bagian **ELSE** akan dijalankan (baris ke-13), dimana saya membuat kalimat "*Tipe data argumen harus berupa angka*" untuk memberitahu pengguna fungsi bahwa tipe argumennya harus berupa angka.

Jika kedua kondisi **is_numeric()** dan **is_int()** benar, maka saya membuat proses **perulangan for** untuk mencari hasil pemangkatan. Setelah hasilnya ditemukan, perintah **return** akan mengembalikan nilai tersebut (baris ke-11).

Dari hasil pemanggilan fungsi, kita dapat melihat bahwa logika alur program sudah berjalan benar, dan jika saya memberikan nilai argumen yang salah, hasil yang ditampilkan bukan kode error PHP, melainkan pesan kesalahan yang lebih informatif.

Dengan menggunakan fungsi seperti **is_numeric()** dan **is_int()** kita dapat melakukan pengecekan tipe data terlebih dahulu sebelum melakukan proses fungsi. Hal ini akan menghindari error program PHP, dan memberikan fleksibilitas untuk melakukan tindakan pencegahan jika tipe data yang diinput bukan yang seharusnya.

4.5 PARAMETER DEFAULT

Default Parameter adalah istilah untuk parameter yang memiliki nilai awal, atau nilai default. Sebagai contoh, misalkan kita membuat fungsi sederhana: **tambah()**. Fungsi ini membutuhkan 2 buah parameter, yakni nilai yang ingin ditambahkan. Berikut adalah contoh programnya:

```
<?php
function tambah($a,$b)
{
return $a+$b;
}
echo tambah(2,3); // hasil: 5
echo tambah(3,4); // hasil: 7
?>
```

Dengan menambahkan default parameter, kita bisa memanggil fungsi tambah() hanya dengan 1 inputan angka, atau bahkan tidak perlu sama sekali. Berikut adalah perubahannya:

```
<?php
function tambah($a,$b=2)
{
return $a+$b;
}
echo tambah(2); // hasil: 4
echo tambah(10); // hasil: 12
echo tambah(4,4); // hasil: 8
?>
```

Perhatikan di dalam pembuatan fungsi, saya menulis parameter kedua menjadi **\$b=2**. Inilah yang dimaksud dengan default parameter. Jika parameter \$b tidak ditulis pada saat pemanggilan fungsi, nilai 2 akan dijadikan nilai awal.

Dengan nilai *default* ini, kita bisa merancang fungsi dengan parameter yang bersifat **opsional**. Parameter tersebut bisa diisi pada saat pemanggilan fungsi, namun boleh juga diabaikan. Sehingga jika fungsi dipanggil tanpa parameter, nilai ini akan menjadi nilai awal untuk fungsi tersebut.

Fitur **default parameter** bisa dimanfaatkan untuk membuat fungsi yang fleksibel, karena pada saat pemanggilan fungsi kita tidak harus menginputkan seluruh parameter, tetapi apa yang dianggap perlu saja.

4.5.1 CARA PENULISAN DEFAULT PARAMETER DALAM PHP

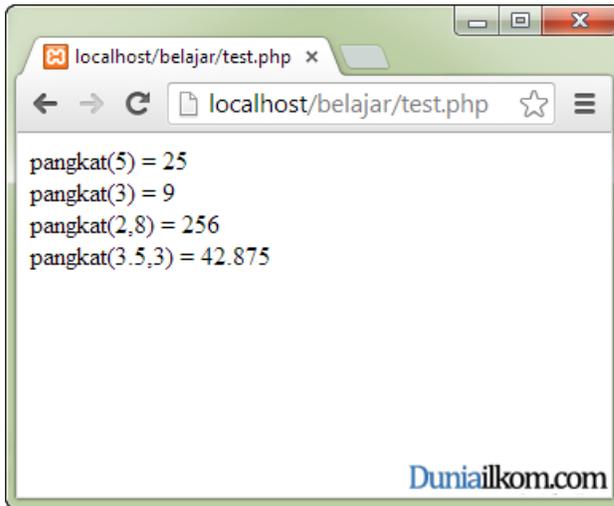
Untuk membuat **default parameter**, kita hanya butuh memberikan nilai awal pada saat pendefinisian parameter. Berikut adalah format dasar penulisan **default parameter** dalam PHP:

```
function nama_fungsi ($parameter1=nilai_default1,$parameter2=nilai_default2)
{
// proses fungsi
return nilai_akhir;
}
```

- **nama_fungsi** adalah nama dari fungsi yang akan dibuat
- **\$parameter1, \$parameter2** adalah variabel yang akan menampung inputan atau argumen pada saat pemanggilan fungsi.
- **nilai_default1, nilai_default2** adalah nilai default untuk *parameter*. Nilai ini akan digunakan jika pada saat pemanggilan fungsi nilai parameter tidak diisi.
- **return nilai_akhir** adalah instruksi untuk mengembalikan hasil pemrosesan fungsi.

Agar lebih mudah memahami konsep **default parameter**, saya akan mengubah fungsi **pangkat()** yang telah kita rancang pada tutorial Cara Pengecekan Tipe Data Argumen untuk Fungsi PHP, berikut adalah contoh kode programnya:

```
<?php
function pangkat($nilai, $pangkat=2)
{
    $hasil=1;
    for ($i=1;$i<=$pangkat;$i++)
    {
        $hasil=$hasil*$nilai;
    }
    return $hasil;
}
//Test beberapa kasus inputan untuk fungsi pangkat()
echo "pangkat(5) = ".pangkat(5);
echo "<br />";
echo "pangkat(3) = ".pangkat(3);
echo "<br />";
echo "pangkat(2,8) = ".pangkat(2,8);
echo "<br />";
echo "pangkat(3.5,3) = ".pangkat(3.5,3);
echo "<br />";
?>
```



Pada contoh fungsi **pangkat()** kali ini saya tidak menggunakan fitur pengecekan tipe data agar contoh program lebih sederhana. Pada saat mendefinisikan fungsi **pangkat()**, saya menambahkan nilai **2** sebagai default parameter untuk parameter **\$pangkat**. Sehingga jika fungsi **pangkat()** dipanggil tanpa parameter ke-2, berarti **\$pangkat** akan diisi nilai **2**.

Saat pemanggilan fungsi **pangkat(5)**, maka kita hanya menggunakan 1 buah argumen. Untuk argumen ke 2, akan diisi nilai default, yakni **2**. Sehingga fungsi yang akan dijalankan sebenarnya adalah **pangkat(5,2)**. Namun saat pemanggilan fungsi **pangkat(2,8)**, maka nilai parameter **\$pangkat** akan menggunakan **8**, bukan angka **2**, karena pada pemanggilan kali ini, saya membuat 2 buah argumen.

4.5.2 PENEMPATAN DEFAULT PARAMETER

Sebuah fungsi tidak dibatasi berapa banyak **default parameter** yang boleh digunakan, namun jika anda ingin membuat **default parameter**, dan pada fungsi yang sama juga menggunakan parameter biasa, maka **default parameter** harus diletakkan di akhir pendefinisian parameter.

Dengan kata lain, setelah pendefinisian parameter dengan nilai default, tidak boleh ada parameter reguler sesudahnya. Contoh pendefinisian fungsi berikut ini akan menyebabkan error dalam PHP:

```
function tambah($nilai1, $nilai2=3, $nilai3)
```

Hal ini terjadi karena default parameter diletakkan sebelum parameter biasa. Contoh diatas seharusnya di tulis sebagai:

```
function tambah($nilai1, $nilai2, $nilai3=3)
//atau
function tambah($nilai1, $nilai2=3, $nilai3=3)
```

Dalam kedua contoh tersebut, **default parameter** di letakkan setelah pendefenisian parameter biasa. **Default parameter** merupakan fitur yang bisa dimanfaatkan untuk membuat fungsi kita lebih fleksibel, karena cara pemanggilan fungsi dapat dirancang dengan lebih sederhana. Untuk merancang fungsi yang lebih rumit, kita bisa membuat jumlah argumen yang tidak terbatas. Mengenai hal ini kita akan membahasnya dalam tutorial Pengertian Variable Parameter dalam Fungsi PHP.

4.6 VARIABEL PARAMETER

Variable Parameter adalah sebuah fitur dalam PHP dimana kita bisa membuat fungsi dengan jumlah **parameter** yang bisa berubah-ubah (*variable*). Umumnya sebuah fungsi membutuhkan **parameter** yang telah ditentukan sebelumnya, namun dengan beberapa fungsi khusus, PHP membolehkan kita untuk membuat fungsi dengan jumlah **parameter** tidak dibatasi, bisa 0, 2, 5, bahkan 100 parameter dengan 1 nama fungsi.

4.6.1 CARA PEMBUATAN FUNGSI DENGAN VARIABLE PARAMETER

Sebuah fungsi dengan jumlah **parameter** yang tidak diketahui tampaknya agak aneh, namun fleksibilitas ini dapat digunakan untuk kasus-kasus pemograman khusus.

Sebagai contoh, saya akan membuat fungsi **penambahan()**, dimana fungsi ini akan menambahkan seluruh angka yang terdapat di dalam **argumennya**. Misalkan **penambahan(2,6,8)** akan menghasilkan **16**, dan **penambahan(1,2,3,4,5,6)** akan menghasilkan nilai **21**. Saya menginginkan fungsi ini mendukung berapapun jumlah argumen. Fungsi akan menggunakan fitur **Variable Parameter**. Untuk membuat sebuah fungsi dengan jumlah parameter yang tidak diketahui, PHP menyediakan 3 fungsi tambahan untuk mengakses argumen yang diinput pada saat fungsi dipanggil. Ketiga fungsi tersebut adalah:

- **func_get_args()**: fungsi ini akan mengembalikan seluruh nilai argumen dalam sebuah fungsi. Hasilnya dalam bentuk **array**.
- **func_num_args()**: fungsi ini akan mengembalikan banyaknya jumlah argumen dalam pemanggilan fungsi, apakah 1 argumen, 3 argumen, atau 10 argumen.
- **func_get_arg(no_urut_argumen)**: fungsi ini akan mengembalikan nilai dari argumen pada nomor urut yang diberikan kepadanya.

Agar mudah memahami fungsi ketiganya, langsung saja kita masuk ke dalam kode program:

```
<?php
function penambahan()
{
```

```

//ambil variable parameter
$array_argumen = func_get_args();
$jumlah_argumen = func_num_args();
$nilai_argumen_ke_2 = func_get_arg(1); //index dimulai dari 0
//tampilkan hasil variable parameter
echo "Jumlah argumen: $jumlah_argumen";
echo "<br />";
echo "Nilai argumen ke-2: $nilai_argumen_ke_2";
echo "<br />";
echo "Array Argumen: ";
var_dump($array_argumen);
echo "<br />";
echo "<br />";
return;
}
echo "Rincian dari fungsi penambahan(1,2) : ";
echo "<br />";
penambahan(1,2);
echo "Rincian dari fungsi penambahan(5,4,3,2,1) : ";
echo "<br />";
penambahan(5,4,3,2,1);
echo "Rincian dari fungsi penambahan(0,6,8,19) : ";
echo "<br />";
echo penambahan(0,6,8,19);
?>

```

Mari kita

membahas kode PHP diatas:

Pada baris ke-2, saya mendefenisikan fungsi **penambahan()** tanpa menggunakan parameter. Untuk membuat fungsi **variable parameter** (dimana jumlah parameternya yang tidak ditentukan) dalam pendefenisian fungsi, dibuat tanpa parameter sama sekali.

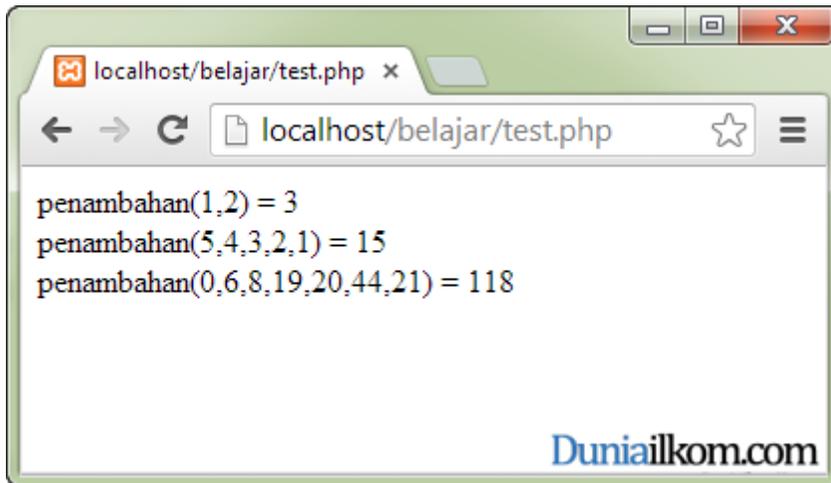
Pad baris 5-7, saya menjalankan ke-3 fungsi khusus yang telah dijelaskan sebelumnya. Fungsi-fungsi ini akan mengambil nilai-nilai dari argumen yang diinputkan pada saat pemanggilan fungsi. Lalu nilai ini saya simpan kedalam 3 variabel, yakni **\$array_argumen**, **\$jumlah_argumen**, dan **\$nilai_argumen_ke_2**. Sebagai catatan, untuk mengambil nilai argumen ke-2, saya dapatkan nilainya dari fungsi **func_get_arg(1)**. Karena argumen dihitung dari angka 0, sehingga argumen kedua berada di index ke 1.

Selanjutnya dari baris 11-20 saya menampilkan hasil masing-masing variabel. Penulisan **echo "
"** digunakan semata-mata agar tampilan di browser lebih rapi dan mudah dilihat.

Pada saat pemanggilan fungsi **penambahan()** pada baris ke 26, 30 dan 34, saya memanggilnya dengan jumlah argumen yang berbeda-beda, dan anda dapat melihat hasil dari ke-3 fungsi khusus variabel parameter.

Dengan ke-3 fungsi khusus telah sukses di jalankan, untuk membuat fungsi **penambahan()** yang sebenarnya, saya tinggal membuat perulangan (looping) untuk menambahkan seluruh argumen-argumen yang ada. Berikut adalah kode program fungsi penambahan versi final:

```
<?php
function penambahan()
{
//ambil variable parameter
$jumlah_argumen = func_num_args();
//proses penambahan
$nilai=0;
for ($i = 0; $i < $jumlah_argumen; $i++)
{
$nilai += func_get_arg($i);
}
return $nilai;
}
echo "penambahan(1,2) = ".penambahan(1,2);
echo "<br />";
echo "penambahan(5,4,3,2,1) = ".penambahan(5,4,3,2,1);
echo "<br />";
echo "penambahan(0,6,8,19,20,44,21) = ".penambahan(0,6,8,19,20,44,21);
?>
```



Fungsi **penambahan()** diatas akan memproses tiap-tiap argumen yang diberikan kepada fungsi tersebut. Perulangan for akan memproses perulangan sebanyak argumen yang dimiliki.

4.7 FUNGSI BAWAAN PHP

4.7.1 CASE CONVERSION

4.7.1.1 MENGUBAH HURUF BESAR KE KECIL

Untuk mengubah huruf besar ke kecil dalam sebuah string PHP, kita bisa menggunakan fungsi **strtolower()**. Fungsi ini membutuhkan 1 buah argumen, yakni string yang akan diubah.

Berikut contoh penggunaan fungsi **strtolower()**:

```
<?php
$kalimat = "SAYA Sedang Belajar PHP di DUNIAILKOM";
$kalimat_new = strtolower($kalimat);
echo $kalimat_new;
// saya sedang belajar php di duniailkom
?>
```

Fungsi **strtolower()** sering digunakan dalam operasi perbandingan string, karena terkadang kita tidak bisa menentukan apakah variabel asal sudah dalam huruf kecil atau dalam huruf besar. Seperti contoh berikut:

```
<?php
$dari_user = "Andi";
```

```

$dari_database = "andi";

if ($dari_user == $dari_database) {
echo "Sama";
}
else {
echo "Tidak Sama";
}

// Hasil: Tidak Sama
?>

```

Kode diatas akan meberikan hasil: **“Tidak Sama”**. Jika kita ingin mengabaikan perbedaan huruf ini, bisa menkonversi terlebih dahulu kedua kata tersebut menggunakan fungsi **strtolower()**:

```

<?php
$dari_user = "Andi";
$dari_database = "andi";

if (strtolower($dari_user) == strtolower($dari_database)) {
echo "Sama";
}
else {
echo "Tidak Sama";
}

// Hasil: Sama
?>

```

Kali ini hasil akhirnya adalah: **“Sama”**. Dalam kebanyakan kasus, sebuah hasil inputan dari user (biasanya berasal dari form), bisa dikonversi terlebih dahulu ke dalam huruf kecil, kemudian baru di simpan ke dalam database, biasanya ini diperlukan untuk pemrosesan **username**:

```

<?php
$user_name = strtolower($_GET["user_name"]);
// proses variabel user_name disini ?>

```

4.7.2 MENGUBAH HURUF KECIL KE BESAR

Untuk mengubah huruf besar ke kecil di dalam PHP, bisa menggunakan fungsi **strtoupper()**. Cara penggunaannya hampir sama dengan fungsi **strtolower()**. Berikut adalah contoh penggunaan fungsi **strtoupper()** di dalam PHP:

```

<?php
$kalimat = "SAYA Sedang Belajar PHP di DUNIAILKOM";
$kalimat_new = strtoupper($kalimat);
echo $kalimat_new;
// SAYA SEDANG BELAJAR PHP DI DUNIAILKOM
?>

```

4.7.3 MENGUBAH HURUF PERTAMA AWAL STRING

Fungsi **ucfirst()** berfungsi untuk mengubah huruf pertama awal string menjadi huruf besar. Langsung saja kita lihat menggunakan contoh kode program:

```
<?php
$kalimat = "saya sedang belajar PHP di Duniailkom";
$kalimat_new = ucfirst($kalimat);
echo $kalimat_new;
// Saya sedang belajar PHP di Duniailkom
?>
```

Perhatikan bahwa fungsi **ucfirst()** 'tidak peduli' dengan karakter lain selain karakter pertama. Dalam contoh diatas kata **"PHP"** menggunakan huruf besar, dan tidak akan terpengaruh oleh fungsi ini.

Selain itu, fungsi **ucfirst()** juga hanya mengubah karakter pertama string, bukan karakter pertama setiap kalimat (yang dipisah dengan tanda titik), seperti kasus berikut ini:

```
<?php
$kalimat = "jangan diganggu! saya lagi serius belajar PHP. di duniailkom.";
$kalimat_new = ucfirst($kalimat);
echo $kalimat_new;
// Jangan diganggu! saya lagi serius belajar PHP. di duniailkom.
?>
```

Secara 'teknis', string diatas terdiri dari 3 kalimat, namun fungsi **ucfirst()** hanya 'melihat' karakter pertama string saja. Jika anda ingin mengubah huruf pertama setiap kalimat, harus membuat fungsi tersendiri.

4.7.4 MENGUBAH HURUF PERTAMA SETIAP KATA

Apabila yang diinginkan adalah agar huruf pertama dalam setiap kata menjadi huruf besar, PHP menyediakan fungsi **ucwords()**. Berikut contoh penggunaannya:

```
<?php
$kalimat = "saya sedang belajar php di duniailkom";
$kalimat_new = ucwords($kalimat);
echo $kalimat_new;
// Saya Sedang Belajar Php Di Duniailkom
?>
```

Seperti yang terlihat, hasil akhirnya setiap huruf pertama setiap kata diubah menjadi huruf besar. Bagaimana jika string tersebut memiliki kombinasi huruf yang tidak sama? Mari kita coba:

```
<?php
$kalimat = "SAYA Sedang Belajar PHP di DUNIAILKOM";
$kalimat_new = ucwords($kalimat);
echo $kalimat_new;
// SAYA Sedang Belajar PHP Di DUNIAILKOM
?>
```

Hasilnya, fungsi **ucwords()** hanya fokus dengan huruf pertama setiap kata. Untuk huruf kedua dan seterusnya, fungsi ini tidak akan melakukan perubahan apapun. Jika anda ingin agar semua karakter ‘seragam’ dimana huruf pertama setiap kata dalam huruf besar, dan kata lain dalam huruf kecil, kita bisa mengkombinasikan fungsi **strtolower()** dengan fungsi **ucwords()**, seperti berikut ini:

```
<?php
$kalimat = "SAYA Sedang Belajar PHP di DUNIAILKOM";
$kalimat_kecil = strtolower($kalimat);
$kalimat_new = ucwords($kalimat_kecil);
echo $kalimat_new;
// Saya Sedang Belajar Php Di Duniailkom
?>
```

Hasilnya, seluruh string akan ditampilkan seragam, tidak peduli bagaimana kombinasi penulisan string awal.

4.7.5 SUBSTR

4.7.5.1 MENGENAL FUNGSI SUBSTR()

Fungsi **substr()** adalah fungsi PHP untuk memotong string, atau untuk mengambil sebagian nilai dari sebuah string. Fitur ini cukup sering digunakan dalam proses pembuatan program PHP, terutama yang membutuhkan manipulasi string.

Sebagai contoh, misalkan kita memiliki sebuah string berbentuk tanggal: “14-09-2015”. Bagaimana caranya untuk mengambil nilai bulan dari string tersebut, yakni karakter “09” ?

Contoh lain, katakan NIM seorang mahasiswa terdiri dari 8 digit: “12140001”. Dua digit pertama adalah tahun masuk mahasiswa, dua digit berikutnya kode jurusan, dan empat digit terakhir adalah no urut mahasiswa. Bagaimana cara memisahkan digit-digit ini?

Dalam kasus seperti inilah fungsi **substr()** diperlukan.

Fungsi **substr()** membutuhkan 2 buah argumen dan 1 argumen tambahan (opsional). Arguman pertama adalah string asal yang ingin diambil nilainya. Argumen kedua berupa posisi awal pemotongan, dan argumen ketiga diisi jumlah karakter yang akan diambil. Argumen kedua dan ketiga bertipe integer dan bisa positif maupun negatif.

PHP membuat penggunaan fungsi **substr()** dengan 6 kombinasi cara penulisan. Kita akan membahasnya satu persatu.

4.7.6 CARA MENGAMBIL KARAKTER DARI AWAL STRING

Penggunaan pertama fungsi **substr()** yang akan kita bahas adalah cara mengambil karakter yang dimulai dari awal string. Berikut contoh penggunaannya:

```
<?php
$kalimat = "Belajar PHP di Duniailkom";
$sub_kalimat = substr($kalimat,3);
echo $sub_kalimat;
// ajar PHP di Duniailkom
?>
```

Dalam kode diatas, saya mengambil string **\$kalimat** mulai dari index ke-3. Jika fungsi **substr()** ditulis dengan 2 argumen seperti ini, dan argumen kedua bernilai positif, maka fungsi **substr()** akan mengembalikan nilai string **\$kalimat** mulai dari huruf ke-4, yakni huruf "a" hingga akhir string.

Perlu menjadi catatan bahwa index string di dalam PHP dimulai dari angka 0. Sehingga fungsi **substr(\$kalimat,3)** akan mengembalikan nilai string **\$kalimat** mulai dari huruf ke-4, dan bukan huruf ke-3. Agar lebih yakin, anda bisa mencoba kode berikut ini:

```
<?php
$kalimat = "123456789";
$sub_kalimat = substr($kalimat,3);
echo $sub_kalimat;
// 456789
?>
```

Fungsi **substr()** juga memiliki argumen ketiga yang bersifat opsional (boleh diisi atau dikosongkan). Jika kita menambahkan argumen ketiga, nilai ini berfungsi sebagai penentu 'berapa banyak jumlah karakter yang akan diambil'. Berikut contohnya:

```
<?php
$kalimat = "Belajar PHP di Duniailkom";
$sub_kalimat = substr($kalimat,8,3);
echo $sub_kalimat;
// PHP
?>
```

Fungsi **substr(\$kalimat,8,3)** akan mengambil string **\$kalimat** mulai dari index ke-8 (karakter ke-9) dan ambil sebanyak 3 karakter. Kita juga bisa memberikan nilai negatif untuk argumen ke-3 ini, dan fungsinya akan berubah. Berikut contohnya:

```
<?php
$kalimat = "Belajar PHP di Duniailkom";
$sub_kalimat = substr($kalimat,8,-3);
```

```
echo $sub_kalimat;
// PHP di Duniaail
?>
```

Fungsi **substr(\$kalimat,8,-3)** akan mengembalikan string **\$kalimat** mulai dari index ke-8 (karakter ke-9) hingga akhir string, kecuali 3 karakter terakhir. 3 karakter terakhir ini adalah “kom”, sehingga hasil akhir kode diatas adalah: “PHP di Duniaail”. Contoh lainnya, apabila kita ingin mengambil string **\$kalimat** mulai dari index ke-10 hingga akhir string, kecuali 5 karakter terakhir, maka fungsinya adalah: **substr(\$kalimat,10,-5)**.

4.7.7 CARA MENGAMBIL KARAKTER DARI AKHIR STRING

Selain dari awal string, kita juga bisa mengambil karakter mulai dari akhir string. Caranya adalah dengan memberikan nilai negatif pada argumen kedua fungsi **substr()**. Langsung saja kita lihat contoh penggunaannya:

```
<?php
$kalimat = "Belajar PHP di Duniaailkom";
$sub_kalimat = substr($kalimat,-10);
echo $sub_kalimat;
// Duniaailkom
?>
```

Fungsi **substr(\$kalimat,-10)** berarti ambil 10 karakter terakhir dari string **\$kalimat**. Agar lebih spesifik, kita juga bisa menentukan jumlah karakter yang ingin diambil. Ini bisa didapat dengan menambahkan argumen ke-3:

```
<?php
$kalimat = "Belajar PHP di Duniaailkom";
$sub_kalimat = substr($kalimat,-10,5);
echo $sub_kalimat;
// Dunia
?>
```

Kombinasi terakhir dari fungsi **substr()** adalah menggunakan angka minus untuk argumen ketiga, seperti contoh berikut:

```
<?php
$kalimat = "Belajar PHP di Duniaailkom";
$sub_kalimat = substr($kalimat,-10,-3);
echo $sub_kalimat;
// Duniaail
?>
```

Fungsi **substr(\$kalimat,-10,-3)** berarti ambil 10 karakter terakhir dari string **\$kalimat**, kecuali 3 karakter terakhir, sehingga hasil kode programnya adalah: “Duniaail”. Dari beberapa contoh diatas, kita sudah melihat 6 jenis kombinasi penggunaan fungsi **substr()**. Kombinasi tersebut di dapat dari

perbedaan jumlah argumen dan apakah argumen tersebut bertipe positif dan negatif. Sebagai kesimpulan, berikut contoh kode seluruh kombinasi fungsi **substr()**:

```
<?php
$kalimat = "Belajar PHP di Duniailkom";
echo substr($kalimat,8); // PHP di Duniailkom
echo "<br>";
echo substr($kalimat,8,6); // PHP di
echo "<br>";
echo substr($kalimat,8,-5); // PHP di Dunia
echo "<br>";
echo substr($kalimat,-10); // Duniailkom
echo "<br>";
echo substr($kalimat,-10,5); // Dunia
echo "<br>";
echo substr($kalimat,-10,-3); // Duniail
?>
```

4.7.8 TRIM

Secara default bawaan PHP, fungsi **trim()** digunakan untuk menghapus spasi atau karakter whitespace dari sebuah string. Karakter spasi yang akan dihapus bisa berada di awal maupun di akhir string.

Dalam prakteknya, fungsi **trim()** sering digunakan untuk 'membersihkan' hasil input form dari karakter spasi yang sengaja atau tidak sengaja ditambahkan pengguna. Berikut adalah contoh penggunaan dasar fungsi trim() di dalam PHP:

```
<?php
$nama = " andi ";
$trim_nama = trim($nama);
echo $trim_nama; // "andi"
?>
```

Karena di HTML whitespace atau spasi tidak akan ditampilkan, penerapan kode diatas tidak terlalu jelas efeknya. Fungsi trim() akan lebih terlihat jika digunakan dalam operasi perbandingan, seperti contoh berikut:

```
<?php
$nama = "andi ";
$nama_juga = "andi";
if ($nama == $nama_juga) {
echo "Nama Sama";
}
else {
echo "Nama Beda";
}
// hasil: Nama Beda
?>
```

Dalam operasi perbandingan diatas, tambahan sebuah spasi diakhir variabel **\$nama**, yakni “andi ” akan membuat operasi perbandingan menghasilkan nilai **FALSE**, sehingga hasil akhirnya adalah “**Tidak Sama**”.

Dengan menambahkan fungsi **trim()**, kode program diatas akan menghasilkan nilai **TRUE**, karena spasi yang ada baik diawal dan diakhir string akan dihapus terlebih dahulu:

```
<?php
$name = "andi ";
$name_juga = "andi";
if (trim($name) == trim($name_juga)) {
echo "Nama Sama";
}
else {
echo "Nama Beda";
}
// hasil: Nama Sama
?>
```

Selain menghapus karakter spasi, fungsi **trim()** juga akan menghapus 5 karakter whitespace lainnya, seperti tab, NEW LINE, CARRIAGE RETURN (karakter enter), NULL-BYTE, dan VERTICAL TAB.

Dalam kode karakter **ASCII**, ke-6 karakter ini adalah sebagai berikut:

- " " (ASCII 32 (0x20)), : karakter spasi.
- "\t" (ASCII 9 (0x09)), : karakter tab.
- "\n" (ASCII 10 (0x0A)), : karakter new line (line feed).
- "\r" (ASCII 13 (0x0D)), : karakter carriage return.
- "\0" (ASCII 0 (0x00)), : karakter NULL-byte.
- "\x0B" (ASCII 11 (0x0B)), : karakter vertical tab.

Berikut contoh penggunaannya:

```
<?php
$name = "\t \t andi \n \r";
$name_juga = " \t andi \t";
if (trim($name) == trim($name_juga)) {
echo "Nama Sama";
}
else {
echo "Nama Beda";
}
// hasil: Nama Sama
?>
```

Untuk menulis karakter “tab”, tidak tersedia tombol khusus di dalam keyboard, oleh karena itu kita menggunakan ESCAPE KARAKTER untuk tab, yakni “\t”.

Fungsi **trim()** ini sering digunakan untuk menfilter hasil imputan form. Berikut adalah contoh penggunaannya:

```
<?php
$nama = trim($_GET["nama"]);
// proses variabel nama disini
?>
```

4.7.9 MENAMBAHKAN KARAKTER YANG AKAN DIHAPUS (CHARACTER_MASK)

Fungsi **trim()** juga memiliki argumen kedua yang bersifat opsional. Argumen kedua ini bertipe string yang jika ditulis akan ditambahkan kedalam daftar karakter yang ikut dihapus, atau istilah teknisnya disebut dengan `character_mask`. Langsung saja kita lihat contoh penggunaannya:

```
<?php
$nama = "__andi__";
$trim_nama = trim($nama);
echo $trim_nama; // "__andi__"
echo "<br>";

$trim_nama = trim($nama, "_");
echo $trim_nama; // "andi"
?>
```

Dengan membuat fungsi **trim(\$nama, "_")** maka karakter underscore “_” juga akan ikut dihapus.

Lebih jauh lagi, **character_mask** ini mendukung penulisan range atau jangkauan karakter, yang ditulis dengan “**awal..akhir**”.

Sebagai contoh, jika saya memiliki beberapa string yang dimulai dengan angka seperti: “1 kelereng”, “2 buah”, “3 orang”, saya bisa menggunakan fungsi trim() untuk menghapus seluruh angka awalan ini. Berikut contohnya:

```
<?php
$kata = "1 kelereng";
$trim_kata = trim($kata, "0..9");
echo $trim_kata; // "kelereng"
echo "<br>";

$kata = "2 buah";
$trim_kata = trim($kata, "0..9");
echo $trim_kata; // "buah"
echo "<br>";

$kata = "3 orang";
$trim_kata = trim($kata, "0..9");
echo $trim_kata; // "orang"
?>
```

Fungsi **trim(\$kata, "0..9")** berarti: hapus whitespace yang ada di awal dan akhir string \$kata, dan hapus karakter 0, 1, 2, 3 s/d 9 di awal dan diakhir string. Fitur untuk menghapus karakter tertentu ini cukup berguna ketika kita ingin ‘membersihkan’ sebuah string dari karakter yang tidak diinginkan.

4.7.9.1 PENGERTIAN FUNGSI RTRIM() DAN LTRIM()

Fungsi **rtrim()** dan **ltrim()** adalah bentuk lain dari fungsi **trim()**, tapi hanya akan menghapus karakter whitespace yang ada disisi kanan (untuk **rtrim()**) dan di sisi kiri string (untuk **ltrim()**). Kedua fungsi ini juga bisa ditambahkan argumen ketiga seperti halnya fungsi **trim()**.

Berikut contoh penggunaan fungsi **rtrim()** dan **ltrim()** dalam PHP:

```
<?php
$nama = " andi ";

$rtrim_nama = rtrim($nama);
echo $rtrim_nama; // " andi"
echo "<br>";

$ltrim_nama = ltrim($nama);
echo $ltrim_nama; // "andi "
echo "<br>";

$nama = "__ andi__";

$rtrim_nama = rtrim($nama, "_");
echo $rtrim_nama; // "__ andi"
echo "<br>";

$ltrim_nama = ltrim($nama, "_");
echo $ltrim_nama; // "andi__"
?>
```

Dapat terlihat bahwa fungsi **rtrim()** dan fungsi **ltrim()** hanya akan menghapus karakter pada satu sisi saja.

4.7.10 EXPLODE

Fungsi **explode()** dalam PHP digunakan untuk menkonversi string menjadi array. Dalam memecah sebuah string menjadi array, fungsi **explode()** membutuhkan beberapa argumen. Berikut adalah format dasar dari fungsi **explode()** dalam PHP:

```
array explode (string $delimiter , string $string [, int $limit ] )
```

Penjelasan dari format diatas adalah:

Nilai kembalian fungsi **explode()** berupa sebuah **array**. Argumen pertama (**\$delimiter**) diisi dengan string karakter yang digunakan sebagai ‘pemecah’ string menjadi array. Argumen kedua (**\$string**) diisi dengan string yang akan dikonversikan.

Argumen ketiga yang bersifat opsional (**\$limit**) diisi dengan batasan jumlah array yang ingin dihasilkan. Argumen ketiga ini bertipe angka (integer) dan bisa berupa nilai positif atau negatif.

Sebagai contoh, dalam kode program berikut ini saya memecah string **\$kalimat** = "satu dua tiga empat lima" menjadi array:

```
<?php
$kalimat = "satu dua tiga empat lima";
$arr_kalimat = explode (" ", $kalimat);
var_dump ($arr_kalimat);

// array(5) {
// [0]=> string(4) "satu" [1]=> string(3) "dua"
// [2]=> string(4) "tiga" [3]=> string(5) "empat"
// [4]=> string(4) "lima"
// }
?>
```

Dalam string **\$kalimat** diatas, saya menggunakan 'spasi' sebagai pemisah kata sehingga untuk memecah string tersebut, spasi ini akan menjadi pembatas (*delimiter*) untuk fungsi `explode`. Jika string awal dipisahkan dengan karakter koma, titik, atau yang lain, kita bisa menggunakannya sebagai karakter delimiter, seperti contoh berikut:

```
<?php
$kalimat = "satu, dua, tiga, empat, lima";
$arr_kalimat = explode ("", $kalimat);
var_dump ($arr_kalimat);

// array(5) {
// [0]=> string(4) "satu" [1]=> string(3) "dua"
// [2]=> string(4) "tiga" [3]=> string(5) "empat"
// [4]=> string(4) "lima"
// }
?>
```

Kali ini string **\$kalimat** dipisah dengan tanda 'spasi dan koma', sehingga saya tinggal menulis fungsi `explode ("", $kalimat)`.

Argumen ketiga fungsi `explode()` bersifat opsional. Argumen ini bisa diisi dengan angka (integer) positif maupun negatif. Jika diisi dengan angka positif, maka angka disini menunjukkan jumlah batasan element array yang akan dihasilkan. Langsung saja kita praktek menggunakan contoh:

```
<?php
$kalimat = "satu, dua, tiga, empat, lima";
$arr_kalimat = explode ("", $kalimat, 3);
var_dump ($arr_kalimat);

// array(3) {
// [0]=> string(4) "satu"
// [1]=> string(3) "dua"
// [2]=> string(17) "tiga, empat, lima"
// }
?>
```

```
?>
```

Dapat anda perhatikan bahwa walaupun string **\$kalimat** seharusnya bisa dipecah menjadi 5 element array, tapi hasil dari fungsi **explode (" ", \$kalimat, 3)** hanya menghasilkan 3 element array, sesuai dengan angka "3" dari argumen ketiga. Untuk element ke-3 array, PHP akan mengambil seluruh string hingga akhir. Namun jika kita menginput nilai negatif, hasilnya akan berbeda:

```
<?php
$kalimat = "satu, dua, tiga, empat, lima";
$arr_kalimat = explode(" ", $kalimat, -3);
var_dump ($arr_kalimat);

// array(2) {
// [0]=> string(4) "satu"
// [1]=> string(3) "dua"
// }
?>
```

Kali ini hanya 2 element array yang dihasilkan. Ini terjadi karena angka "-3" berarti: *kurangi total element array sebanyak 3*. Apabila saya menggunakan fungsi explode 'biasa', akan terdapat 5 element array yang bisa dihasilkan (seperti contoh pertama dalam tutorial ini), namun karena saya menambahkan nilai -3 pada argumen ke 3, hasil akhirnya akan dikurangi sebanyak 3 element (5 - 3 = 2 element).

Fungsi **number_format()** adalah fungsi bawaan PHP yang bisa digunakan untuk memformat tampilan angka, baik itu angka *integer* maupun *float*. Dengan memformat tampilan angka, akan membuatnya lebih 'cantik' dan mudah dibaca. Selain itu fungsi ini juga akan membulatkan atau menambahkan angka "0" dibelakang koma jika dibutuhkan.

Fungsi **number_format()** memiliki 2 buah cara penulisan, yakni dengan 2 argumen atau 4 argumen. Untuk fungsi dengan 2 argumen, berikut adalah format dasar penulisannya:

```
string number_format ( float $number [, int $decimals = 0 ] )
```

- Hasil akhir fungsi ini bertipe string.
- Argumen pertama (\$number) membutuhkan input nilai angka yang akan diformat. Argumen ini bertipe float, tapi bisa juga bisa diisi dengan nilai integer.
- Argumen kedua (\$decimals) bersifat opsional. Argumen ini menentukan berapa jumlah angka desimal (angka di belakang koma) yang dibutuhkan. Apabila tidak diisi, dianggap sebagai 0.

Langsung saja kita lihat contoh penggunaan fungsi **number_format()** ini:

```
<?php
$angka = 1999.12345;
$angka_format = number_format($angka);
```

```
echo $angka_format;
// 1,999
?>
```

Dapat terlihat, fungsi **number_format()** akan “membuang” seluruh bagian desimal dari angka diatas. Ini terjadi karena secara default, fungsi ini menggunakan 0 sebagai jumlah digit desimal (angka di belakang koma). Sebenarnya fungsi **number_format()** tidak benar-benar “membuang” nilai desimal, tetapi membulatkannya, seperti contoh berikut:

```
<?php
$angka = 1999.99;
$angka_format = number_format($angka);
echo $angka_format;
// 2,000
?>
```

Dapat terlihat bahwa nilai 1999.99 akan dibulatkan menjadi 2,000. Dengan menambahkan argumen kedua, kita bisa menentukan berapa digit desimal yang diperlukan:

```
<?php
$angka = 1999.888;
$angka_format = number_format($angka,2);
echo $angka_format;
// 1,999.89
?>
```

Kali ini fungsi **number_format()** akan membatasi 2 digit desimal. Selain itu dapat anda lihat bahwa fungsi ini juga membulatkan nilai desimalnya. Bagaimana jika angka awal tidak memiliki angka desimal? Mari kita lihat:

```
<?php
$angka = 1999;
$angka_format = number_format($angka,3);
echo $angka_format;
// 1,999.000
?>
```

Hasilnya, string akhir akan ditambahkan angka “0” pada bagian desimalnya. Dengan demikian kita bisa membuat format angka dengan panjang seragam. Perhatikan juga bahwa PHP tetap menggunakan tanda koma “,” sebagai pemisah nilai ribuan, dan titik “.” sebagai pemisah nilai desimal. Ini adalah aturan penulisan di Amerika. Nantinya, dengan fungsi **number_format()** kita juga bisa mengubah format ini agar sesuai dengan aturan penulisan angka di Indonesia, dimana tanda titik digunakan sebagai pemisah ribuan, dan tanda koma sebagai pemisah nilai desimal. Bentuk penulisan lain dari fungsi **number_format()** membutuhkan 4 argumen. Berikut format dasar penulisannya:

```
string number_format ( float $number , int $decimals = 0 ,
string $dec_point = "." , string $thousands_sep = "," )
```

Format penulisan diatas terlihat rumit, namun sebenarnya cukup sederhana. Argumen ketiga dan keempat fungsi **number_format()** digunakan untuk menentukan karakter apa sebagai pemisah nilai ribuan dan nilai desimal. Sebagai contoh, karena di Indonesia kita menggunakan karakter titik sebagai pemisah angka ribuan dan karakter koma sebagai pemisah desimal, saya bisa memformat angka tersebut dengan fungsi **number_format()**, seperti contoh berikut:

```
<?php
$angka = 1999.12345;
$angka_format = number_format($angka,2,"",".");
echo $angka_format;
// 1.999,12
?>
```

Dapat terlihat sekarang, angka akan ditampilkan dalam format 'umum' yang kita gunakan sehari-hari. Lebih jauh lagi, kita bisa menambahkan string "**Rp.**" sebagai awalan agar hasilnya pas untuk nilai mata uang rupiah. Berikut contohnya:

```
<?php
$angka = 3050145.756;
$angka_format = number_format($angka,2,"",".");
echo "Rp. ".$angka_format;
// Rp. 3.050.145,76
?>
```

Dalam kode diatas saya menyambungkan awal "**Rp.**" dengan string hasil fungsi **number_format()**. Tampilan tersebut cocok digunakan untuk hasil akhir angka mata uang untuk membuat laporan yang banyak melibatkan nilai uang.

4.7.11 STRPOS

Fungsi **strpos()** adalah fungsi bawaan PHP yang bisa digunakan untuk mencari posisi sebuah karakter atau sebuah string di dalam string lainnya. Hasil akhir fungsi ini adalah angka yang menunjukkan posisi karakter/string yang ingin dicari. Berikut format dasar penulisan fungsi **strpos()** di dalam PHP:

```
mixed strpos (string $haystack , mixed $needle [, int $offset = 0 ] )
```

- Hasil akhir fungsi **strpos** adalah **mixed**, karena bisa berupa angka (posisi ditemukannya karakter yang ingin dicari), maupun **FALSE** jika fungsi ini tidak menemukan nilai apa-apa.
- Argumen pertama (**\$haystack**) berupa string asal dimana pencarian akan dilakukan. String ini bisa terdiri dari sebuah kalimat pendek hingga dokumen string panjang yang terdiri dari ribuan huruf.

- Argumen kedua (**\$needle**) diisi dengan karakter yang akan dicari. Karakter disini bisa berupa sebuah huruf/angka atau string lain seperti kalimat.
- Argumen ketiga (**\$offset**) bersifat opsional, digunakan untuk ‘melompati’ hasil pencarian dari posisi tertentu. Ini digunakan untuk mengabaikan beberapa kalimat awal pencarian.

Agar lebih mudah dipahami, langsung saja kita membahas contoh penggunaannya:

```
<?php
$kalimat="Sedang serius belajar PHP di duniailkom";
$posisi=strpos($kalimat,"e");
echo $posisi;
// 1
?>
```

Hasil angka 1 dari kode program diatas menunjukkan karakter ‘e’ pertama dalam string **\$kalimat**. Perlu diingat bahwa di dalam PHP penomoran index string dimulai dari angka 0. Sehingga posisi 1 diatas sebenarnya berada pada karakter ke-2. Selain menggunakan 1 karakter, kita juga bisa mencari sebuah kata atau string lain, seperti contoh berikut:

```
<?php
$kalimat="Sedang serius belajar PHP di duniailkom";
$cari="serius";
$posisi=strpos($kalimat,$cari);
echo $posisi;
// 7
?>
```

Dalam kode program diatas, saya menempatkan string “**serius**” di dalam sebuah variabel, dan mencari posisinya dengan fungsi strpos(). Hasilnya, kata “serius” di temukan dan berada pada index ke-7. Bagaimana jika di dalam string tidak ditemukan apa-apa? Mari kita coba:

```
<?php
$kalimat="Sedang serius belajar PHP di duniailkom";
$posisi=strpos($kalimat,"CSS",9);
var_dump($posisi);
// bool(false)
?>
```

Hasilnya: **FALSE**. Saya sengaja menggunakan fungsi **var_dump()** karena jika menggunakan echo, tidak akan terdapat tampilan apa-apa. Ini terjadi karena perintah echo mengkonversi boolean **FALSE** menjadi string kosong dan menampilkannya. Hasil **FALSE** ini bisa kita manfaatkan untuk membuat sebuah kondisi pemeriksaan apakah karakter yang ingin dicari ada di dalam string, seperti contoh berikut:

```
<?php
$kalimat="Sedang serius belajar PHP di duniailkom";
$posisi=strpos($kalimat,"CSS");
if ($posisi){
echo "Ketemu";
```

```

}
else {
echo "Tidak ketemu";
}
// Tidak ketemu
?>

```

Dengan membuat kondisi seperti itu, kita bisa membuat kode program untuk menentukan ada tidaknya sebuah karakter. Akan tetapi, efek '[konversi tipe data otomatis](#)' dari PHP bisa menimbulkan masalah tersendiri. Perhatikan kode berikut ini:

```

<?php
$kalimat="Sedang serius belajar PHP di duniailkom";
$posisi=strpos($kalimat,"Sedang");
if ($posisi){
echo "Ketemu";
}
else {
echo "Tidak ketemu";
}
// Tidak ketemu
?>

```

Hasilnya adalah: **"Tidak ketemu"** ! Padahal seperti yang kita lihat di dalam string asal, kata "Sedang" ada di posisi pertama string. Apa yang terjadi? Ini karena efek '[type juggling](#)' dimana PHP akan mengkonversi sebuah tipe data ke tipe data lain secara otomatis. Hasil dari fungsi `strpos($kalimat,"Sedang")` adalah 0, karena kata "Sedang" ditemukan di awal string, dan indexnya adalah 0. Namun sewaktu kita menggunakan hasil ini dalam kondisi IF, nilai 0 akan dikonversi menjadi **FALSE**, sehingga hasilnya adalah: **"Tidak ketemu"**. Agar kode program kita bebas dari masalah ini, bisa mengubah kondisi pemeriksaan IF menjadi seperti berikut ini:

```

<?php
$kalimat="Sedang serius belajar PHP di duniailkom";
$posisi=strpos($kalimat,"Sedang");
if ($posisi !== FALSE){
echo "Ketemu";
}
else {
echo "Tidak ketemu";
}
// Ketemu
?>

```

Kali ini saya memaksa PHP untuk memeriksa apakah nilai variabel `$posisi` harus bertipe boolean **FALSE**. Kesalahan program seperti ini sering terjadi dan sangat susah untuk ditemukan, karena PHP tidak akan menampilkan pesan error apapun. Kembali ke fungsi `strpos()`, kali ini saya akan mencoba menambahkan argumen ke-3:

```

<?php
$kalimat="Sedang serius belajar PHP di duniailkom";

```

```
$posisi=strpos($kalimat,"e",9);
echo $posisi;
// 15
?>
```

Argumen ketiga dari fungsi `strpos()` berfungsi untuk ‘melompati’ beberapa karakter awal. Dalam contoh diatas, karakter “e” yang ingin dicari sebenarnya sudah ditemukan pada index ke-1 dan ke-8. Namun karena saya menambahkan argumen ke-3 dengan nilai 9, maka proses pencarian baru dilakukan mulai dari index ke-9.

4.7.12 PENGERTIAN FUNGSI STRIPOS()

Fungsi `strpos()` yang kita bahas disini bersifat **case sensitif**, yang berarti akan membedakan antara huruf besar dan huruf kecil. Sebagai contoh, kode program berikut akan menghasilkan nilai **FALSE**:

```
<?php
$kalimat="Sedang serius belajar PHP di duniailkom";
$posisi=strpos($kalimat,"php",9);
var_dump($posisi);
// bool(false)
?>
```

Ini terjadi karena fungsi `strpos()` akan membedakan antara “php” dengan “PHP”. Kedua kata ini akan dianggap berbeda, sehingga karakter “php” tidak ditemukan. Jika kita ingin untuk mengabaikan perbedaan ini, bisa menggunakan fungsi `stripos()`. Penambahan huruf ‘i’ disini menunjukkan bahwa fungsi `stripos()` bersifat **case-insensitive**. Yang berarti akan mengabaikan perbedaan huruf besar dan kecil. Fungsi `stripos()` juga mendukung argumen yang sama seperti fungsi `strpos()`.

Berikut adalah hasil kode program diatas jika menggunakan fungsi `stripos()`:

```
<?php
$kalimat="Sedang serius belajar PHP di duniailkom";
$posisi=stripos($kalimat,"php",9);
echo $posisi;
// 22
?>
```

Kali ini karakter ‘php’ ditemukan pada posisi index ke-22, dimana kata “PHP” berada.

4.7.13 IMPLODE

Fungsi `implode()` adalah fungsi bawaan PHP untuk menggabungkan array menjadi string. Berikut format dasar penulisan fungsi `implode()`:

```
string implode ( string $glue , array $pieces )
```

- Hasil akhir fungsi `implode` berupa string.

- Argumen pertama diisi dengan string (\$glue) yang berfungsi sebagai karakter penyatu array.
- Argumen kedua diisi dengan array (\$pieces), yakni array yang akan digabungkan.

Berikut contoh penggunaannya:

```
<?php
$arr_kata = ["satu", "dua", "tiga", "empat", "lima"];
$kalimat = implode(" ", $arr_kata);
echo $kalimat;
// satu dua tiga empat lima
?>
```

Pada kode diatas, variabel **\$arr_kata** berisi array dengan 5 element, kemudian saya menggabungkannya menjadi string menggunakan fungsi **implode(" ", \$arr_kata)**. Hasilnya, setiap element array akan digabung dengan spasi. Kita juga bisa menggunakan karakter lain untuk penggabungan ini, seperti contoh berikut:

```
<?php
$arr_kata = ["satu", "dua", "tiga", "empat", "lima"];
$kalimat = implode(", ", $arr_kata);
echo $kalimat;
// satu, dua, tiga, empat, lima
?>

<br>

<?php
$arr_kata = ["satu", "dua", "tiga", "empat", "lima"];
$kalimat = implode(" | ", $arr_kata);
echo $kalimat;
// satu | dua | tiga | empat | lima
?>
```

Dalam dua contoh diatas, saya menggunakan karakter “,” dan “|” sebagai pemisah string. Perhatikan juga bahwa fungsi implode() tidak menambahkan karakter tersebut di awal dan akhir string, tapi hanya diantara element array. Karena alasan historis, PHP juga membolehkan penulisan fungsi implode() hanya dengan 1 argumen, yakni array yang akan digabung:

```
<?php
$arr_kata = ["satu", "dua", "tiga", "empat", "lima"];
$kalimat = implode($arr_kata);
echo $kalimat;
// satuduatigaempatlima
?>
```

Fungsi **implode()** diatas sebenarnya sama dengan fungsi **implode("", \$arr_kata)**, dimana kita menggunakan string kosong untuk menggabungkan array. Hasil akhirnya, seluruh string digabung tanpa

ada pemisah (menjadi 1 string panjang). Namun agar konsisten dengan fungsi **explode()**, sebaiknya kita tetap menggunakan 2 argumen untuk fungsi **implode()** ini.

5 KONEKSI DATABASE MYSQL

PHP dan MySQL adalah kombinasi bahasa pemrograman dan database yang sangat populer. Situs duniaikom yang berbasis wordpress juga menggunakan kombinasi PHP dan MySQL.

Dalam bagian tutorial belajar PHP lanjutan ini, saya akan membuat tutorial mengenai cara membuat koneksi antara PHP dengan database MySQL.

Untuk dapat mengikuti tutorial ini, saya sarankan anda telah mengetahui dasar penulisan PHP, seperti perulangan, tipe data, dan pemanggilan fungsi, karena kita akan banyak menggunakannya dalam tutorial kali ini. Pengetahuan tentang query MySQL juga akan sangat membantu.

Jika anda belum paham keduanya, atau untuk sekedar me-refresh tentang PHP dan MySQL, duniaikom telah membuat tutorial lengkap tentang keduanya. Untuk Tutorial PHP Dasar, dapat anda baca di Tutorial Belajar PHP Dasar Untuk Pemula dan untuk Tutorial MySQL Dasar, dapat anda baca di Tutorial Belajar MySQL Dasar Untuk Pemula.

5.1 JENIS-JENIS KONEKSI KE MYSQL

PHP merupakan bahasa pemrograman yang selalu di update dan berkembang mengikuti teknologi terbaru. Saat ini, **Pemrograman Berorientasi Objek** (OBJECT ORIENTED PROGRAMMING) merupakan trend pemrograman PHP, dan hal ini juga mempengaruhi cara mengakses database MySQL dari PHP.

PHP memiliki 3 cara pengaksesan MySQL, yakni melalui **PDO (PHP Data Objects)**, **mysqli extension** dan **mysql extension**. **PDO** menggunakan PEMROGRAMAN OBJEK, **mysqli extension** tersedia dalam bentuk OBJEK dan PROSEDURAL (diakses melalui fungsi-fungsi) sedangkan **mysql extension** sepenuhnya menggunakan pemrograman PROSEDURAL.

Mari kita bahas pengertian dan perbedaan ketiga metode ini:

5.1.1 KONEKSI MYSQL DENGAN MYSQL EXTENSION

Saat pertama kali mempelajari PHP MySQL sekitar tahun 2008 (atau jika anda pernah mempelajari PHP-MySQL beberapa tahun yang lalu), untuk mengakses MySQL dari PHP, kita menggunakan fungsi-fungsi seperti `MYSQL_CONNECT()`, `MYSQL_QUERY()`, dan `MYSQL_FETCH_ARRAY()`. Fungsi-fungsi ini tergabung ke dalam **mysql extension** (saat itu **PDO** dan **mysqli extension** masih jarang digunakan)

Namun sekarang (tepatnya mulai PHP versi 5.5.0) PHP memutuskan untuk membuat **mysql extension** berstatus DEPRECATED. Yang artinya pengaksesan database MySQL menggunakan fungsi MYSQL EXTENSION sudah tidak disarankan lagi. Programmer PHP diharapkan pindah ke **mysqli extension** atau **PDO** yang berbasis objek. Alasannya, MySQL versi terbaru memiliki fitur-fitur yang semakin lengkap dan kompleks, sehingga PHP memutuskan untuk membuat fungsi extension baru agar programmer PHP bisa menggunakan fitur-fitur ini.

5.1.2 KONEKSI MYSQL DENGAN MYSQLI EXTENSION

Sebagai pengganti **mysql extension**, PHP menyediakan **mysqli extension** (**mysqli** merupakan singkatan dari **MySQL Improved**). Mysqli extension ini pada dasarnya adalah perbaikan dari MYSQL EXTENSION dan dikembangkan untuk mendukung fitur-fitur terbaru untuk **MySQL 4.1** keatas.

Hampir semua fungsi yang ada pada MYSQL EXTENSION juga tersedia pada MYSQLI. Syntax (aturan penulisan) **mysqli** sangat mirip dengan **mysql extension**. Sehingga jika anda telah lama menggunakan **mysql extension**, akan sangat mudah untuk beralih menggunakan **mysqli extension**.

Selain menggunakan **mysql** maupun **mysqli extension**, cara ketiga untuk pengaksesan database MySQL dari PHP adalah dengan menggunakan PHP Data Objects (atau sering disingkat dengan **PDO**).

5.1.3 KONEKSI MYSQL DENGAN PDO (PHP DATA OBJECTS)

PDO (PHP Data Objects), adalah EXTENSION atau penambahan fitur dalam PHP yang dirancang sebagai INTERFACE UNIVERSAL untuk pengaksesan berbagai jenis database (tidak hanya MySQL). Contohnya, jika kita menggunakan **PDO** dalam menulis kode pemrograman, lalu suatu saat website kita bertukar database dari **MySQL** ke **Oracle**, maka kita tidak perlu mengubah semua kode program, cukup mengubah cara pemanggilan **PDO** diawal program saja. Dari ketiga cara koneksi PHP dengan MySQL ini, metode yang disarankan adalah menggunakan **mysqli** atau **PDO**.

5.1.4 PERBANDINGAN ANTARA MYSQLI, PDO DAN MYSQL EXTENSION

Sebagai perbandingan fitur antara ketiga jenis koneksi PHP-MySQL ini, berikut adalah tabel perbandingan antara **mysqli**, **PDO** dan **mysql extension** yang bersumber dari manual resmi PHP:

Comparison of MySQL API options for PHP			
	PHP's mysqli Extension	PDO (Using PDO MySQL Driver and MySQL Native Driver)	PHP's MySQL Extension
PHP version introduced	5.0	5.0	Prior to 3.0
Included with PHP 5.x	yes	yes	Yes
MySQL development status	Active development	Active development as of PHP 5.3	Maintenance only
Recommended by MySQL for new projects	Yes - preferred option	Yes	No
API supports Charsets	Yes	Yes	No
API supports server-side Prepared Statements	Yes	Yes	No
API supports client-side Prepared Statements	No	Yes	No
API supports Stored Procedures	Yes	Yes	No
API supports Multiple Statements	Yes	Most	No
Supports all MySQL 4.1+ functionality	Yes	Most	No

5.1.5 CARA PENULISAN MYSQL EXTENSION, MYSQLI EXTENSION, DAN PDO (PHP DATA OBJECTS)

Untuk mengetahui secara sekilas perbedaan cara pengaksesan database MySQL menggunakan **mysql extension**, **mysqli extension**, dan **PDO**, berikut contoh kode PHPnya:

```
<?php
// cara mengakses MySQL menggunakan mysql extension:
$link = mysql_connect("localhost", "root", "qwerty");
mysql_select_db("universitas");
$result = mysql_query("SELECT * FROM mahasiswa");
$row = mysql_fetch_assoc($result);

// cara mengakses MySQL menggunakan mysqli extension:
$mysqli = new mysqli("localhost", "root", "qwerty", "universitas");
$result = $mysqli->query("SELECT * FROM mahasiswa");
$row = $result->fetch_assoc();

// cara mengakses MySQL menggunakan PDO:
$pdo = new PDO('mysql:host=localhost;dbname=universitas', 'root', 'qwerty');
$stmt = $pdo->query("SELECT * FROM mahasiswa");
```

```
$row = $statement->fetch(PDO::FETCH_ASSOC);  
?>
```

Dalam contoh diatas, dianggap bahwa user MySQL adalah **root**, password '**qwerty**', dan nama database '**universitas**'. Anda tidak perlu memahami kode program diatas, karena kita akan membahasnya dengan lengkap dalam tutorial belajar PHP MySQL di duniaikom ini. Untuk tahap pertama, saya akan menggunakan metode **mysql extension** yang berbasis fungsi terlebih dahulu. Walaupun metode ini tidak disarankan lagi, namun metode inilah yang paling mudah dipelajari dan telah dikenal luas. Setelah selesai membahas **mysql extension**, selanjutnya kita akan beralih ke **mysqli extension** yang bisa ditulis dengan PROCEDURAL STYLE (menggunakan fungsi-fungsi) ataupun dengan OBJECT STYLE. Terakhir, kita akan membahas cara membuat koneksi PHP MySQL menggunakan PDO. Cara pengaksesan MySQL dengan PDO yang berbasis objek akan lebih mudah dipahami jika anda telah mempelajari [pemrograman berbasis objek](#).

5.2 MENGAKTIFKAN PDO

PDO (*PHP Data Objects*) adalah interface universal yang disediakan PHP untuk “berkomunikasi” dengan database server. Maksud istilah “*interface universal*” disini adalah bahwa PDO tidak terikat dengan aplikasi database tertentu. Apabila saat ini kita menggunakan database **MySQL** dan dikemudian hari ingin bermigrasi menggunakan **PostgreSQL**, kita hanya tinggal mengganti cara pemanggilan awal PDO dan seluruh kode program yang ada bisa langsung digunakan untuk database baru.

Kondisi ini berbeda jika menggunakan **mysql extension** atau **mysqli extension**. Kedua extension ini hanya bisa bekerja dengan database MySQL. Karena alasan inilah banyak sebaiknya kita mulai beralih menggunakan PDO dibandingkan *mysqli*.

Secara teknis, apabila menggunakan *mysql* atau *mysqli extension*, PHP langsung berhubungan dengan MySQL Server, tetapi jika menggunakan PDO, ia tidak berhubungan langsung dengan database, tetapi hanya sebagai “interface”. Konsep PDO ini dapat digambarkan sebagai berikut:

PHP PDO -> Database Driver -> Database Server

PDO bekerja dengan metode yang disebut “**data-access abstraction layer**”. Artinya, apapun jenis database server yang digunakan, kode PHP yang ditulis akan tetap sama. PDO menyediakan “*abstraction layer*” untuk berkomunikasi dengan database server.

5.2.1 JENIS DATABASE SERVER YANG DIDUKUNG PDO

Hingga saat penulisan tutorial ini (PHP versi 5.6) PDO mendukung setidaknya 12 jenis Interface/Database Server:

- CUBRID (PDO)
- MS SQL Server (PDO)
- Firebird (PDO)
- IBM (PDO)
- Informix (PDO)
- MySQL (PDO)
- MS SQL Server (PDO)
- Oracle (PDO)
- ODBC and DB2 (PDO)
- PostgreSQL (PDO)
- SQLite (PDO)
- 4D (PDO)

List ini dapat anda lihat dari <http://php.net/manual/en/pdo.drivers.php>.

Dapat dilihat bahwa PDO mendukung banyak aplikasi database populer seperti: ORACLE, MICROSOFT SQL SERVER, dan POSTGRESQL. Dengan membuat kode PHP menggunakan PDO, secara tidak langsung kita juga membuka kemungkinan untuk menggunakan database server selain MySQL, sehingga menjadi lebih fleksibel.

Walaupun **PDO Extension** telah aktif secara “DEFAULT” pada PHP versi 5.1 keatas, tetapi tidak semua database driver bisa digunakan. Dengan kata lain, untuk alasan performa, PHP me-nonaktifkan beberapa driver database seperti **Oracle** atau **PostgreSQL** di dalam PDO.

Untuk melihat driver database apa saja yang telah aktif dan bisa langsung digunakan, bisa menggunakan fungsi (lebih tepatnya: STATIC METHOD) **PDO::getAvailableDrivers()**:

```
<?php
print_r(PDO::getAvailableDrivers());
?>
```

Ketika saya menjalankan perintah tersebut pada PHP 5.6 bawaan XAMPP, berikut adalah hasil yang didapat:

```
Array ( [0] => mysql [1] => sqlite )
```

Dari tampilan diatas, dapat dilihat bahwa driver PDO bawaan PHP yang aktif (dan yang bisa digunakan) hanyalah **MySQL** dan **SQLite**. Jadi bagaimana cara mengaktifkan driver untuk database lainnya? Caranya adalah mengubah setingan **php.ini**.

File PHP.INI adalah file konfigurasi yang digunakan PHP. Saya telah membahas sekilas tentang cara menemukan dan mengakses file php.ini dalam tutorial [Cara Mengubah File Konfigurasi PHP \(php.ini\)](#). Karena saya menggunakan PHP yang diinstall dari XAMPP di drive D, lokasinya adalah: **D:\xampp\php\php.ini**.

Silahkan buka file **php.ini** dengan aplikasi text editor, kemudian search kata “**pdo**”. Dalam versi PHP yang saya gunakan, hasil pencarian “pdo” akan tampil pada baris ke-1010, seperti gambar dibawah:

```
1010 extension=php_openssl.dll
1011 ;extension=php_pdo_firebird.dll
1012 extension=php_pdo_mysql.dll
1013 ;extension=php_pdo_oci.dll
1014 ;extension=php_pdo_odbc.dll
1015 ;extension=php_pdo_pgsql.dll
1016 extension=php_pdo_sqlite.dll
1017 ;extension=php_pdo_sqlite_external.dll
1018 ;extension=php_pgsql.dll
1019 ;extension=php_pspell.dll
1020 ;extension=php_shmop.dll
1021
```

Duniailkom.com

Pada baris inilah kita mengaktifkan driver database PDO. Perhatikan library yang diawali dengan “**php_pdo**”, inilah driver-driver database PDO yang tersedia di dalam PHP. Dapat dilihat driver yang telah aktif hanya **php_pdo_mysql.dll** dan **php_pdo_sqlite.dll**.

Untuk mengaktifkannya, silahkan hapus tanda titik koma (;) dari awal baris nama driver yang ingin digunakan. Sebagai contoh, saya akan mengaktifkan **extension=php_pdo_pgsql.dll** yang merupakan driver dari database PostgreSQL:

```
1010 extension=php_openssl.dll
1011 ;extension=php_pdo_firebird.dll
1012 extension=php_pdo_mysql.dll
1013 ;extension=php_pdo_oci.dll
1014 ;extension=php_pdo_odbc.dll
1015 extension=php_pdo_pgsql.dll
1016 extension=php_pdo_sqlite.dll
1017 ;extension=php_pdo_sqlite_external.dll
1018 ;extension=php_pgsql.dll
1019 ;extension=php_pspell.dll
1020 ;extension=php_shmop.dll
1021
```



Duniailkom.com

Selanjutnya, save file PHP.INI, kemudian RESTART web server **Apache** (matikan dan hidupkan kembali melalui XAMPP CONTROL PANEL). Untuk memastikan apakah driver telah aktif atau belum, kita kembali menjalankan method **PDO::getAvailableDrivers()**, dan berikut adalah hasil yang didapat:

```
Array ( [0] => mysql [1] => pgsql [2] => sqlite )
```

Seperti yang terlihat, driver **PostgreSQL** untuk PDO telah aktif dan bisa langsung digunakan.

Pada saat penulisan contoh diatas, saya juga mencoba mengaktifkan **extension=php_pdo_oci.dll** yang digunakan database ORACLE, tetapi mendapati error:

“THE PROGRAM CAN’T START BECAUSE OCI.DLL IS MISSING FROM YOUR COMPUTER”

Error diatas terjadi karena PHP tidak menyertakan library **php_pdo_oci.dll**, dan kita harus mendownloadnya secara manual.

Setelah saya mencari dari beberapa sumber, ternyata PDO untuk oracle masih berstatus “EXPERIMENT”, dan dikembangkan oleh komunitas, bukan Oracle sendiri.

Hal ini bisa menjadi pertimbangan jika anda ingin menggunakan PDO untuk mengakses database Oracle. Sebagai alternatif, silahkan gunakan fungsi oracle “biasa”: **oci8**.

5.3 MEMBUAT KONEKSI MYSQL PDO

Karena **PDO** adalah fungsi/fitur PHP yang ditulis menggunakan objek, maka untuk membuat koneksi antara PHP dengan MySQL, kita menggunakan **PDO constructor**, yakni dengan membuat objek baru dari class PDO (menggunakan keyword: **new**).

Berikut adalah struktur dasar cara penulisan PDO CONSTRUCTOR untuk database MySQL:

```
$dbh = new PDO('mysql:host=lokasi_db;dbname=nama_database',  
"nama_user", "password_user");
```

Variabel **\$dbh** adalah variabel yang akan menjadi **object PDO**. Object ini biasa disebut sebagai **“Database Handler”**. Object inilah yang nantinya akan kita gunakan untuk menjalankan perintah-perintah PDO. Nama variabel boleh bebas, dan tidak harus **\$dbh**.

Argumen dari CONSTRUCTOR PDO terdiri dari 3 bagian, bagian pertama berisi **nama database server**(misalnya:mysql) kemudian diikuti dengan **alamat server** dan **nama database**, semuanya di dalam 1 string. Untuk argumen kedua diisi dengan **nama user**, dan argumen ketiga berisi **password user**.

Sebagai contoh, untuk masuk ke dalam MySQL server di komputer **localhost**, dengan user **root**,password: **12345**, nama database: **universitas**, kode PDO yang digunakan adalah sebagai berikut:

```
<?php  
$dbh = new PDO('mysql:host=localhost;dbname=universitas', "root", "12345");  
?>
```

Setelah membuat koneksi PDO, untuk menghapus koneksi kita tinggal memberikan nilai **“null”** kepada variabel **“Database Handler”** sebagai berikut:

```
<?php  
$dbh = null;  
?>
```

Sama seperti **mysqli**, proses penutupan koneksi ini bersifat opsional, dan PHP secara otomatis akan menghapus koneksi pada saat halaman PHP selesai dijalankan.

Walaupun dalam tutorial ini kita hanya membahas MySQL, sebagai perbandingan berikut adalah cara melakukan koneksi untuk database server **MS SQL, Sybase** dan**SQLite**:

Koneksi PDO untuk **MS SQL Server** Database:

```
$dbh = new PDO("mssql:host=$host;dbname=$dbname, $user, $pass");
```

Koneksi PDO untuk **Sybase** Database:

```
$dbh = new PDO("sybase:host=$host;dbname=$dbname, $user, $pass");
```

Koneksi PDO untuk **SQLite** Database:

```
$dbh = new PDO("sqlite:my/database/path/database.db");
```

5.3.1 CARA MENAMPILKAN PESAN KESALAHAN (ERROR) PADA PDO

Karena PDO menggunakan object, penanganan kesalahan akan lebih baik jika menggunakan blok **TRY...CATCH**, atau dikenal dengan “EXCEPTIONS”.

Untuk menampilkan pesan kesalahan dengan exceptions pada PDO, terdapat 3 pilihan metode. Ketiga metode ini menggunakan method **PDO::setAttribute(PDO::ATTR_ERRMODE)**:

```
$dbh->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_SILENT );
$dbh->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING );
$dbh->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );
```

PDO::ERRMODE_SILENT adalah pilihan default. Jika kita tidak mengubah setingan error, pilihan inilah yang digunakan. Untuk mengecek error, kita bisa menggunakan method **PDO::errorCode()** dan **PDO::errorInfo()**.

PDO::ERRMODE_WARNING digunakan untuk menampilkan error berupa warning PHP, kemudian PHP akan melanjutkan proses eksekusi program. Pilihan ini cocok untuk proses debugging.

PDO::ERRMODE_EXCEPTION digunakan untuk menampilkan error dengan exceptions. Pilihan inilah yang sering digunakan untuk menampilkan error dengan TRY...CATCH.

Dengan menggunakan penanganan kesalahan (EXCEPTIONS), berikut adalah modifikasi kode PHP untuk koneksi PDO dengan MySQL:

```
<?php
try {
// buat koneksi dengan database
$dbh = new PDO('mysql:host=localhost;dbname=universitas', "root", "");

// set error mode
$dbh->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );

// hapus koneksi
$dbh = null;
}
catch (PDOException $e) {
// tampilkan pesan kesalahan jika koneksi gagal
print "Koneksi atau query bermasalah: " . $e->getMessage() . "<br/>";
die();
}
?>
```

Sebagai contoh, jika saya mengganti nama user menjadi “USER_TETANGGA” yang memang tidak ada di dalam database, berikut tampilan error yang terjadi:

```
Koneksi atau query bermasalah: SQLSTATE[HY000] [1044]
Access denied for user '@'localhost' to database 'universitas'
```

Didalam block TRY...CATCH inilah nantinya kita akan menjalankan query-query MySQL seperti menampilkan data, maupun menginput data baru ke dalam tabel MySQL. Untuk cara menampilkan data tabel MySQL menggunakan PDO, akan kita bahas dalam tutorial berikutnya: Cara Menampilkan data MySQL menggunakan PDO (PHP Data Object).

5.4 MENAMPILKAN DATA PDO

Walaupun terkesan “rumit”, menjalankan query dan menampilkan hasilnya menggunakan **PDO PHP** tidaklah sulit. Bahkan sangat mirip dengan [mysqli](#). PHP juga menyediakan beragam cara untuk menampilkan hasil query MySQL dengan PDO.

Sama seperti **mysql extension** dan **mysqli extension**, untuk menjalankan dan menampilkan data dari MySQL dengan PDO, kita membutuhkan 3 langkah: **buat koneksi**, **jalankan query**, dan **tampilkan hasil**.

Agar lebih mudah dipahami, berikut adalah contoh kode program untuk menampilkan seluruh tabel **mahasiswa_ilkom** dengan menggunakan PDO:

```
<?php
try {
    // buat koneksi dengan database
    $dbh = new PDO('mysql:host=localhost;dbname=universitas', "root", "");

    // set error mode
    $dbh->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );

    // jalankan query
    $result = $dbh->query('SELECT * FROM mahasiswa_ilkom');

    // tampilkan data
    while($row = $result->fetch()) {
        echo "$row[0] $row[1] $row[2] $row[3] $row[4]";
        echo "<br />";
    }
    // hapus koneksi
    $dbh = null;
}
catch (PDOException $e) {
    // tampilkan pesan kesalahan jika koneksi gagal
    print "Koneksi atau query bermasalah: " . $e->getMessage() . "<br/>";
    die();
}
?>
```

Dalam contoh diatas, saya menjalankan PDO di dalam block **TRY...CATCH** agar bisa menampilkan pesan kesalahan (ERROR). Selanjutnya, saya membuat koneksi dengan MySQL Server dan menyimpan hasil koneksi ke dalam variabel **\$dbh**. Kita telah membahas tentang hal ini dalam tutorial sebelumnya.

Kode program **\$dbh->setAttribute** digunakan untuk men-set kode error yang dihasilkan agar bisa "DITANGKAP" oleh block **CATCH**.

Untuk menjalankan query dengan PDO, kita bisa menggunakan method **\$dbh->query()**. Method ini membutuhkan 1 argumen bertipe string, yakni query yang akan dijalankan. Dalam contoh diatas, saya menggunakan query **'SELECT * FROM mahasiswa_ilkom'** untuk menampilkan seluruh tabel **mahasiswa_ilkom**.

Method **\$dbh->query()** akan mengembalikan nilai berupa objek baru yang digunakan untuk proses berikutnya. Object ini saya tampung ke dalam variabel **\$result**.

Kemudian, untuk menampilkan hasil query saya menggunakan method **\$result->fetch()**. Method ini akan mengembalikan nilai berupa array yang bisa ditampilkan dengan perulangan while seperti biasa.

Apabila anda menjalankan perintah tersebut, isi tabel **mahasiswa_ilkom** akan ditampilkan di web browser:

```
089023013 Alex Supriyanto 23 Surabaya 2.90
089023020 Naira Alika 20 Padang 3.90
089023023 Alika Shanum 21 Medan 3.80
089023026 Rina Melita 22 Lampung 3.50
089023031 Joni Halim 21 Palembang 3.60
```

5.4.1 MENGUBAH CARA MENAMPILKAN DATA PDO

Jika anda perhatikan, dalam contoh diatas kita mengakses hasil query menggunakan **array** dengan penomoran **index**. Selain dengan INDEX ARRAY, PDO menyediakan berbagai cara untuk menampilkan data sesuai kebutuhan.

Untuk mengubah cara menampilkan data PDO, kita tinggal menambahkan 1 argumen ke dalam method **\$result->fetch()**. Misalkan untuk menampilkan hasil query menggunakan nama kolom sebagai index array, kita bisa menggunakan **PDO::FETCH_ASSOC** sebagai berikut:

```
<?php
```

```
// kode program untuk koneksi dan menjalankan query
// ...
// ...
// tampilkan data
while($row = $result->fetch(PDO::FETCH_ASSOC)) {
    echo $row['nim']." ".$row['nama']." ".$row['umur']." ";
    echo $row['tempat_lahir']." ".$row['IPK'];
    echo "<br />";
}
?>
```

Dan jika ingin menampilkan dengan pengaksesan objek, bisa menggunakan **PDO::FETCH_OBJ**:

```
<?php
// kode program untuk koneksi dan menjalankan query
// ...
// ...
// tampilkan data
while($row = $result->fetch(PDO::FETCH_OBJ)) {
    echo $row->nim." ".$row->nama." ".$row->umur." ";
    echo $row->tempat_lahir." ".$row->IPK;
    echo "<br />";
}
?>
```

Selain dengan settingan **PDO::FETCH_ASSOC** dan **PDO::FETCH_OBJ**, PDO PHP juga menyediakan beberapa pilihan lain:

- **PDO::FETCH_BOTH**: mengembalikan nilai berupa array yang bisa di tampilkan menggunakan index angka maupun index nama kolom. Ini adalah pilihan default jika method **\$result->fetch()** ditulis tanpa argumen.
- **PDO::FETCH_NUM**: mengembalikan nilai berupa array yang bisa di tampilkan dengan index angka.
- **PDO::FETCH_CLASS**: mengembalikan nilai ke dalam class yang telah disiapkan.
- **PDO::FETCH_LAZY**: mengkombinasikan **PDO::FETCH_BOTH** dan **PDO::FETCH_OBJ**, sehingga bisa dipanggil dengan array maupun objek.

5.5 MENGINPUT DATA MYSQL PDO

Secara garis besar, proses menginput data MySQL dengan PDO tidak jauh berbeda dengan cara menampilkan data dengan PDO yang telah kita bahas pada tutorial sebelumnya.

PHP menyediakan banyak cara untuk menginputkan data kedalam MySQL menggunakan PDO extension. Pada tutorial ini kita akan membahas 2 diantaranya, yakni menggunakan method **PDO->query()** dan **PDO->exec()**.

5.5.1.1 MENGINPUT DATA MYSQL DENGAN PDO->QUERY()

Untuk menginput data MySQL dengan method **PDO->query()**, kita tinggal menambahkan query INSERT atau UPDATE kedalam method ini.

Sebagai contoh, saya ingin menambahkan 1 data baru ke dalam tabel **mahasiswa_ilkom**. Berikut adalah contoh kode programnya:

```
<?php
try {
    // buat koneksi dengan database
    $dbh = new PDO('mysql:host=localhost;dbname=universitas', "root", "");

    // set error mode
    $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $query= 'INSERT INTO mahasiswa_ilkom VALUES
    ("099023022", "Ayla Ramadhani", 20, "Aceh", 3.1)';

    // jalankan query
    $result = $dbh->query($query);
    // tampilkan hasil proses query
    echo $result->rowCount()." data berhasil ditambahkan";

    // hapus koneksi
    $dbh = null;
} catch (PDOException $e) {
    // tampilkan pesan kesalahan jika koneksi gagal
    print "koneksi/query bermasalah: " . $e->getMessage() . "<br/>";
    die();
}
?>
```

Sebagian besar FUNCTION dan METHOD diatas sudah kita pelajari dalam beberapa tutorial sebelum ini.

Setelah proses koneksi dan men-set error display, saya membuat sebuah string untuk menampung query INSERT, yakni di dalam variabel **\$query**. Kemudian query dijalankan dengan method **\$dbh->query(\$query)**.

Setelah query di jalankan, kita bisa mengetahui berapa baris tabel MySQL yang ditambahkan menggunakan method **\$result->rowCount()**.

Apabila anda menjalankan perintah diatas, hasil yang ditampilkan adalah:

```
1 data berhasil ditambahkan
```

Sebagai latihan, silahkan anda tambahkan kode program diatas untuk menampilkan data tabel **mahasiswa_ilkom** setelah proses input data.

5.5.1.2 MENGINPUT DATA MYSQL DENGAN PDO->EXEC()

Selain menggunakan method **PDO->query()**, kita juga bisa menggunakan method **PDO->exec()** untuk menjalankan query MySQL. Cara penulisan method **PDO->exec()** juga sangat mirip dengan method **PDO->query()**. Mari kita lihat bagaimana cara menambahkan 1 data ke dalam tabel **mahasiswa_ilkom** menggunakan method **PDO->exec()**:

```
<?php
try {
    // buat koneksi dengan database
    $dbh = new PDO('mysql:host=localhost;dbname=universitas', "root", "");

    // set error mode
    $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $query= 'INSERT INTO mahasiswa_ilkom VALUES
    ("109013027", "Siska Rosiana", 19, "Bengkulu", 2.9)';

    // jalankan query
    $result = $dbh->exec($query);

    // tampilkan hasil proses query
    echo $result." data berhasil ditambahkan";

    // hapus koneksi
    $dbh = null;
} catch (PDOException $e) {
    // tampilkan pesan kesalahan jika koneksi gagal
    print "koneksi/query bermasalah: " . $e->getMessage() . "<br/>";
    die();
}
?>
```

Dari kode diatas, method **\$dbh->exec(\$query)** akan menjalankan query yang disimpan dari variabel **\$query**. Untuk menampilkan jumlah data yang ditambahkan, kita bisa langsung menggunakan variabel **\$result** hasil method **\$dbh->exec(\$query)**. Jika anda menjalankan kode diatas, maka hasil yang didapat adalah sebagai berikut:

```
1 data berhasil ditambahkan
```

5.6 MENGGUNAKAN PREPARED STATEMENT

PHP mendukung fitur **prepared statements** pada **mysqli extension** dan **PDO**. Fitur ini juga umum digunakan di dalam pemrograman PHP lanjutan seperti **FRAMEWORK PHP**. Karena hal ini, penting juga bagi kita untuk memahami cara penggunaannya.

5.6.1 PENGERTIAN PREPARED STATEMENTS MYSQL

Prepared statements adalah sebuah fitur yang disediakan MySQL (dan juga beberapa aplikasi database lainnya), dimana kita bisa mengirim **query** (perintah) secara terpisah antara query inti dengan “data” dari query. Tujuannya, agar query menjadi lebih aman dan cepat (jika perintah yang sama akan digunakan beberapa kali).

Sebagai perbandingan, untuk menampilkan data MySQL menggunakan fungsi **mysqli_query()**, kita membuat seluruh query dalam 1 string dan langsung mengirimkannya ke MySQL Server, sebagai berikut:

```
$result=mysqli_query("SELECT * FROM mahasiswa_ilkom WHERE
nama='Neil Situmorang'");
```

Dengan **prepared statements**, query tersebut akan dipisah antara perintah query: “SELECT...” dengan ‘data’-nya yakni “NEIL SITUMORANG”. Kali ini kita akan lihat bagaimana cara menggunakannya di dalam extension PDO. Langsung saja kita masuk kedalam contoh program. Kali ini saya ingin menambahkan 1 data kedalam tabel **mahasiswa_ilkom** menggunakan *prepared statement* dengan PDO:

```
<?php
try {
    // buat koneksi dengan database
    $dbh = new PDO('mysql:host=localhost;dbname=universitas', "root",
    "");

    // set error mode
    $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // siapkan query (prepare)
    $stmt = $dbh->prepare('INSERT INTO mahasiswa_ilkom
VALUES (?, ?, ?, ?, ?)');

    // hubungkan data dengan variabel (bind)
    $stmt->bindParam(1, $nim_mhs);
    $stmt->bindParam(2, $nama_mhs);
    $stmt->bindParam(3, $umur_mhs);
    $stmt->bindParam(4, $tempat_lahir_mhs);
    $stmt->bindParam(5, $ipk_mhs);
    // siapkan "data" query
    $nim_mhs="109023031";
    $nama_mhs="Jerry Kurniawan";
    $umur_mhs=23;
    $tempat_lahir_mhs="Malang";
    $ipk_mhs=3.1;

    // jalankan query (execute)
    $stmt->execute();

    // tampilkan hasil proses query
    echo $stmt->rowCount()." data berhasil ditambahkan";
```

```

    // hapus koneksi
    $dbh = null;
} catch (PDOException $e) {
    // tampilkan pesan kesalahan jika koneksi gagal
    print "koneksi/query bermasalah: " . $e->getMessage() . "<br/>";
    die();
}
?>

```

Seperti yang pernah kita bahas pada *mysql extension*, untuk menjalankan prepared statement, kita membutuhkan 3 langkah: **prepare**, **bind** dan **execute**. Proses **prepare** dijalankan menggunakan method **\$dbh->prepare()**. Method ini membutuhkan 1 argumen berupa perintah query yang akan dijalankan. Hasil dari method kemudian disimpan kedalam variabel **\$stmt**. Variabel inilah yang akan kita gunakan selama menjalankan proses prepared statement. Proses **bind** dilakukan menggunakan method **\$stmt->bindParam()**. Method ini membutuhkan 2 argumen. Argumen pertama adalah angka urutan dari “tempat data” yang ditulis pada proses *prepare*. Karena pada proses *prepare* saya menuliskan: **INSERT INTO mahasiswa_ilkom VALUES (?, ?, ?, ?, ?)**, maka setiap tanda “?” harus di **bind**. Urutan tanda “?” inilah yang digunakan dalam argumen pertama method **\$stmt->bindParam()**. Untuk argumen kedua method **\$stmt->bindParam()** adalah variabel penampung yang nantinya akan diisi data. Selanjutnya setiap variable yang di defenisikan akan saya input nilai yang sesuai. Setelah seluruh data diinput kedalam variabel, kita tinggal menjalankan method **\$stmt->execute()** untuk menjalankan query prepared statement. Jika anda menjalankan kode program diatas, hasilnya adalah sebagai berikut:

```
1 data berhasil ditambahkan
```

Agar lebih yakin, anda bisa menambahkan kode program diatas untuk menampilkan seluruh isi tabel **mahasiswa_ilkom**. Selain menggunakan tanda “?” untuk menandakan bagian dari data, PDO PHP juga menyediakan cara lain, yakni dengan menuliskan ‘nama untuk data’ atau dikenal dengan **“named parameters prepared statements”**. Berikut adalah revisi contoh program menggunakan *named parameters prepared statement*:

```

<?php
try {
    // buat koneksi dengan database
    $dbh = new PDO('mysql:host=localhost;dbname=universitas', "root",
    "");

    // set error mode
    $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // siapkan query (prepare)
    $stmt = $dbh->prepare('INSERT INTO mahasiswa_ilkom VALUES

```

```

(:nim_mhs, :nama_mhs, :umur_mhs, :tempat_lahir_mhs, :ipk_mhs)');

// hubungkan data dengan variabel (bind)
$stmt->bindParam(':nim_mhs', $nim_mhs);
$stmt->bindParam(':nama_mhs', $nama_mhs);
$stmt->bindParam(':umur_mhs', $umur_mhs);
$stmt->bindParam(':tempat_lahir_mhs', $tempat_lahir_mhs);
$stmt->bindParam(':ipk_mhs', $ipk_mhs);

// siapkan "data" query
$nim_mhs="109023031";
$nama_mhs="Jerry Kurniawan";
$umur_mhs=23;
$tempat_lahir_mhs="Malang";
$ipk_mhs=3.1;

// jalankan query (execute)
$stmt->execute();

// tampilkan hasil proses query
echo $stmt->rowCount()." data berhasil ditambahkan";

// hapus koneksi
$dbh = null;
} catch (PDOException $e) {
// tampilkan pesan kesalahan jika koneksi gagal
print "koneksi/query bermasalah: " . $e->getMessage() . "<br/>";
die();
}
?>

```

Perbedaan dari kode program sebelumnya adalah pada cara penulisan query dan argumen method **bindParam()**. Kali ini kita menggunakan named variable (diawali dengan tanda ":"), seperti **:nim_mhs**, **:nama_mhs**, atau **:umur_mhs**. Named variable ini selanjutnya menjadi argumen pertama method **\$stmt->bindParam()**.

Versi penulisan terakhir *prepared statement* yang akan kita pelajari adalah dengan menggabungkan proses **bind** dan **execute** sekaligus. Agar cara ini bisa dijalankan, data dari prepared statement harus dibuat dalam bentuk array. Berikut contohnya:

```

<?php
try {
// buat koneksi dengan database
$dbh = new PDO('mysql:host=localhost;dbname=universitas', "root",
"");

// set error mode
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

// siapkan query (prepare)
$stmt = $dbh->prepare('INSERT INTO mahasiswa_ilkom VALUES

```

```

(:nim_mhs, :nama_mhs, :umur_mhs, :tempat_lahir_mhs, :ipk_mhs)');

// siapkan "data" query
$nim_mhs="109023031";
$nama_mhs="Jerry Kurniawan";
$umur_mhs=23;
$tempat_lahir_mhs="Malang";
$ipk_mhs=3.1;

// jalankan query (execute)
$stmt->execute(array(':nim_mhs' => $nim_mhs, ':nama_mhs' =>
$nama_mhs,
':umur_mhs' => $umur_mhs, ':tempat_lahir_mhs' => $tempat_lahir_mhs,
':ipk_mhs' => $ipk_mhs));

// tampilkan hasil proses query
echo $stmt->rowCount()." data berhasil ditambahkan";

// hapus koneksi
$dbh = null;
} catch (PDOException $e) {
// tampilkan pesan kesalahan jika koneksi gagal
print "koneksi/query bermasalah: " . $e->getMessage() . "<br/>";
die();
}
?>

```

Perbedaan dalam kode program diatas terletak dalam **\$stmt->execute()**. Kali ini method **\$stmt->execute()** kita input dengan array sebagai argumen yang akan menjadi 'data' untuk query. Dengan cara ini kita tidak perlu menggunakan method **\$stmt->bindParam()**.

6 FORM PHP

6.1 MEMBUAT DAN MEMPROSES FORM HTML DENGAN PHP

Langkah pertama untuk dapat memproses hasil form **HTML**, adalah mempelajari cara 'MENGAMBIL' nilai dari form dan menampilkannya. Dalam tutorial kali ini, kita akan membahas [Cara Menampilkan Hasil Form HTML dengan PHP](#).

6.1.1 MENGENAL STRUKTUR DASAR FORM HTML (ATRIBUT ACTION DAN METHOD)

Tutorial mengenai cara membuat form menggunakan **HTML** telah saya bahas lengkap di [tutorial form HTML](#), dalam tutorial form PHP kali ini kita hanya fokus kepada bagian form yang berkaitan dengan PHP.

Berikut adalah struktur dasar form sederhana dalam HTML:

```

<form action="proses.php" method="get">
Nama: <input type="text" name="nama" />
<br />

```

```
E-Mail: <input type="text" name="email" />  
<br />  
<input type="submit" value="Proses Data" />  
</form>
```

Jika anda menjalankan form HTML tersebut, akan ditampilkan form sederhana dengan 2 buah kotak inputan dan sebuah tombol “PROSES DATA” yang berfungsi untuk **submit** form. Dari struktur dasar tersebut, di dalam tag **<form>** terdapat 2 buah atribut. Yakni atribut **action** dan atribut **method**. Kita akan membahas kedua atribut ini secara lebih rinci.

Atribut pertama adalah **action**. Atribut **action** ini diisi dengan nilai berupa alamat halaman PHP dimana kita akan memproses isi form tersebut. Dalam contoh diatas, saya membuat nilai **action="proses.php"**, yang berarti saya harus menyediakan sebuah file dengan nama: **proses.php** untuk memproses form tersebut.

Isi atribut action sebenarnya adalah alamat dari halaman PHP. Karena atribut ACTION pada contoh diatas ditulis **action="proses.php"**, maka file **proses.php** harus berada di dalam 1 folder dengan halaman HTML yang berisi form ini. Namun anda bisa dengan bebas mengubah alamat **proses.php** ini tergantung dimana file tersebut berada, misalnya menjadi alamat relatif seperti **action="file_php/proses.php"**, ataupun alamat absolut seperti **action="https://www.duniailkom.com/proses.php"**.

Atribut kedua yang berkaitan dengan pemrosesan form HTML adalah atribut **method**. Atribut inilah yang akan menentukan bagaimana cara form ‘DIKIRIM’ ke dalam halaman **proses.php**. Nilai dari atribut **method** hanya bisa diisi dengan 1 dari 2 pilihan, yakni **get** atau **post**.

Jika seperti contoh diatas saya membuat nilai **method="get"**, maka nilai dari form akan dikirim melalui alamat URL WEBSITE. Namun jika nilai method diubah menjadi **method="post"**, maka nilai form tidak akan terlihat di dalam alamat URL. Perbedaan antara **method get** dan **post** akan kita bahas secara mendalam dalam tutorial selanjutnya. Nilai dari atribut **method** ini juga akan mempengaruhi cara kita memproses nilai dari form.

Setelah membuat tag pembuka form dengan atribut **action** dan **method**, isi form selanjutnya adalah 2 buah tag **<input type="text">** yang akan menampilkan kotak isian form. Hal yang paling penting diperhatikan adalah atribut **name** dari masing-masing tag **<input>**. Nilai dari **name** inilah yang menjadi penanda masing-masing objek form agar dapat diproses dengan PHP.

Setelah 2 buah TEXT INPUT, OBJEK FORM terakhir adalah tombol **submit** yang apabila di klik akan mengirimkan data dari form ke halaman **proses.php** untuk diproses. Atribut penting disini adalah atribut **type="submit"**, yang akan otomatis mengirim isian form ketika tombol ini di klik.

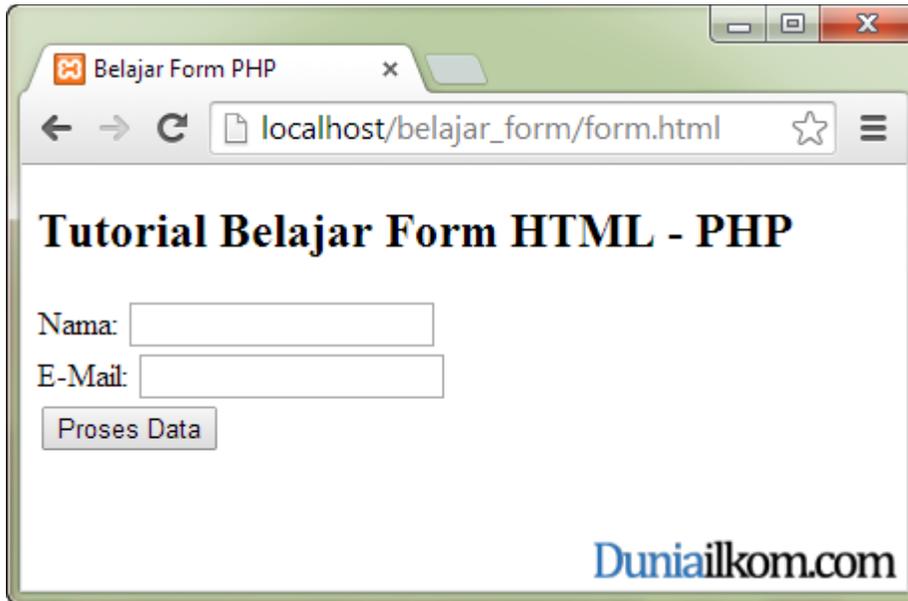
6.1.2 CARA MENGIRIMKAN NILAI FORM HTML KE DALAM PHP

Untuk memahami Cara Mengirimkan Nilai Form HTML ke dalam PHP, kita akan langsung praktek dengan membuat 2 buah file, yakni halaman HTML yang berisi form dengan nama file **form.html**, dan halaman PHP yang akan berisi kode untuk menampilkan hasil form dengan nama file: **proses.php**.

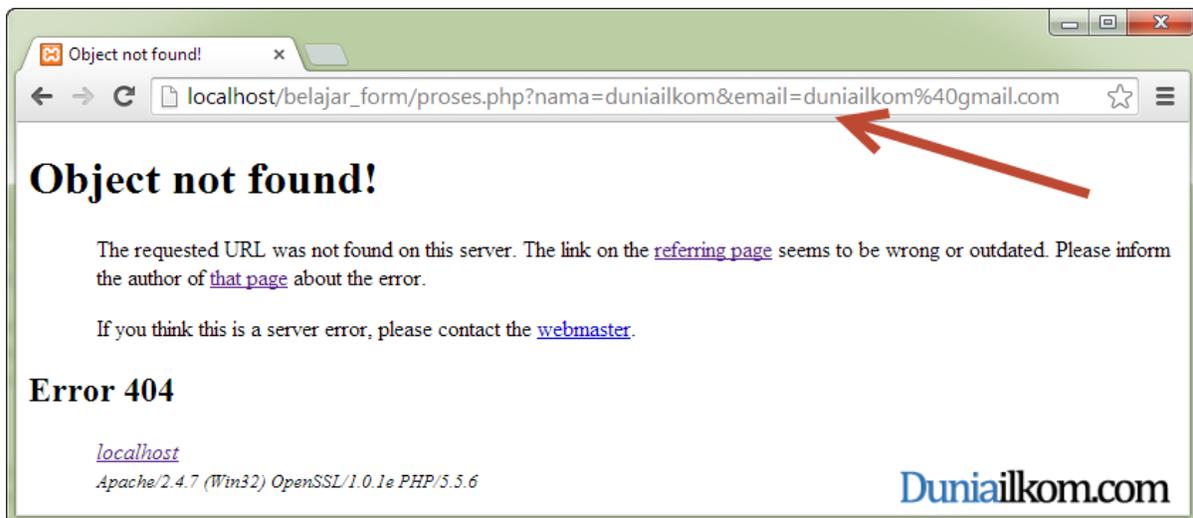
Karena kita akan mengeksekusi kode **PHP**, kedua file ini harus dijalankan dengan **XAMPP** dan berada di dalam folder **htdocs**. Untuk contoh kali ini saya akan membuat folder **belajar_form** di dalam folder **htdocs XAMPP**, sehingga untuk mengakses kedua halaman adalah dari alamat : http://localhost/belajar_form/form.html dan http://localhost/belajar_form/proses.php (mengenai cara install dan menggunakan **XAMPP** telah saya bahas pada tutorial [Cara Menjalankan File PHP dengan XAMPP](#))

Sebagai langkah pertama, kita akan membuat file **form.html** yang berisi kode HTML sebagai berikut:

```
<!DOCTYPE html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Belajar Form PHP</title>
</head>
<body>
<h2>Tutorial Belajar Form HTML - PHP </h2>
<form action="proses.php" method="get">
Nama: <input type="text" name="nama" />
<br />
E-Mail: <input type="text" name="email" />
<br />
<input type="submit" value="Proses Data" />
</form>
</body>
</html>
```



Kode HTML diatas hanya berisi struktur kode HTML sederhana dengan 1 buah form yang berisi 2 text inputan untuk **nama** dan **e-mail**. Struktur form ini persis sama dengan form kita bahas pada bagian awal tutorial ini. Sebelum membuat halaman **proses.php**, kita akan mencoba melakukan sedikit percobaan dengan form HTML ini. Silahkan coba input kedua kotak isian form ini dengan nilai apapun dan klik tombol **submit**. Ketika anda men-klik tombol submit, anda akan mendapati halaman error seperti berikut ini:



Halaman error tersebut memberitahu kita bahwa halaman **proses.php** tidak ditemukan (yang memang belum kita buat), namun perhatikan alamat **URL** pada ADDRESS BAR web browser, kita bisa melihat ada penambahan seperti berikut ini:

http://localhost/belajar_form/proses.php?nama=duniaikom&email=duniaikom%40gmail.com

Perhatikan bahwa alamat URL sekarang telah berganti menjadi **proses.php**, yang kemudian diikuti dengan **?nama=duniailkom&email=duniailkom%40gmail.com**. Pesan inilah yang sebenarnya di kirim oleh halaman **form.html** sewaktu kita men-klik tombol SUBMIT.

Tanda ? menandakan awal dari data form, dan kemudian diikuti dengan nama dari objek form dan nilainya, dalam contoh ini, nilai yang akan dikirim adalah **nama=duniailkom**. Karakter “&” digunakan sebagai tanda pemisah nilai objek form yang 1 dengan yang lain, kemudian diikuti dengan nilai kedua **email=duniailkom%40gmail.com**. Tanda %40 dalam kode ini adalah kode karakter **HTML** untuk karakter @ yang biasa digunakan di dalam email.

Jika anda menambahkan objek form ketiga, maka alamat URL akan semakin panjang. Kita bisa melihat data yang dikirim karena pada saat pembuatan form, saya menggunakan **method=get**. Namun jika anda merubah form HTML kita dengan menggunakan **method=post**, maka anda tidak akan melihat karakter-karakter ini di dalam URL.

Selanjutnya kita akan mencoba menampilkan nilai ini dengan PHP pada halaman **proses.php**.

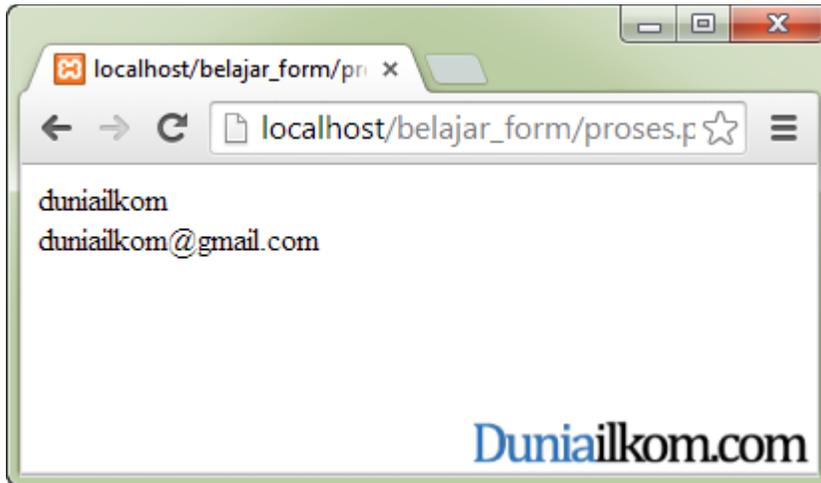
6.1.3 CARA MENAMPILKAN NILAI FORM HTML DENGAN PHP (\$_GET DAN \$_POST)

Setelah membuat halaman **form.html** yang berisi form HTML, kita akan membuat halaman **proses.php** yang berisi kode PHP untuk menangani nilai dari form ini.

Silahkan buat file **proses.php** dengan kode program sebagai berikut, dan savelah di dalam folder yang sama dengan **form.html** berada:

```
<?php
echo $_GET['nama'];
echo "<br />";
echo $_GET['email'];
?>
```

Sebelum kita membahas kode program PHP tersebut, silahkan buka kembali halaman **form.html**, isi kotak input NAMA dan EMAIL, lalu klik tombol PROSES DATA. Apabila tidak ada error, maka akan tampil hasil berikut ini:



Tampilan diatas adalah hasil dari 3 baris kode program PHP yang kita buat di dalam halaman **proses.php**.

Untuk mengambil nilai form HTML, PHP menyediakan 2 buah variabel global yaitu variabel **\$_GET** dan **\$_POST**. Kita menggunakan variabel **\$_GET** jika pada saat pembuatan form menggunakan atribut **method=get**, dan menggunakan variabel **\$_POST** jika form dibuat dengan **method=post**.

Kedua variabel ini sebenarnya adalah array, sehingga cara mengakses nilai dari form adalah dengan cara: **\$_GET['nama_objek_form']**.

'**NAMA_OBJEK_FORM**' adalah nilai dari atribut **name** di dalam **form**. Jika kita memiliki tag dengan kode HTML **<input type="text" name="nama" />**, maka untuk mengakses nilainya adalah dengan **\$_GET['nama']**, dan untuk tag **<input type="text" name="email" />** diakses dengan nilai **\$_GET['email']**.

Sebagai latihan, silahkan anda mengganti atribut **METHOD** dalam file **form.html** menjadi:

```
<form action="proses.php" method="post">
```

Lalu ubah juga file **proses.php** menjadi:

```
<?php  
echo $_POST['nama'];  
echo "<br />";  
echo $_POST['email'];  
?>
```

Dan PHP akan menampilkan hasil yang sama, namun kali ini form dikirim menggunakan **method=post**.

Di dalam tutorial kali ini kita telah membahas dasar [Cara Menampilkan Hasil Form HTML dengan PHP](#), namun apa yang kita bahas disini hanya cara paling dasar untuk menampilkan nilai form dengan PHP. Dalam tutorial Form PHP selanjutnya kita akan membahas lebih dalam tentang perbedaan pengiriman form dengan method GET dan POST.

6.2 METODE GET DAN POST

6.2.1 KEUNGGULAN DAN KELEMAHAN METHOD FORM 'GET'

Kelemahan yang paling jelas jika kita menggunakan **method=get** adalah nilai dari form dapat dilihat langsung di dalam **URL** yang dikirimkan. Jika anda membuat **form** untuk data-data yang sensitif seperti **PASSWORD**, maka form dengan **method=get** bukanlah pilihan yang tepat.

Form dengan **method=get** disarankan untuk form yang berfungsi menampilkan data, yaitu dimana hasil isian form hanya digunakan untuk menampilkan data, sesuai dengan arti kata **get** yang bisa berarti: 'AMBIL'. Sehingga **method=get** sebaiknya digunakan untuk form yang 'MENGAMBIL' data dari database.

Salah satu penggunaan **method=get** yang umum digunakan adalah pada form pencarian (FORM SEARCH). Dengan membuat hasil inputan form terlihat jelas dalam **URL**, pengunjung web bisa dengan mudah menebak 'ALUR KERJA' dari situs kita. Bahkan pengunjung bisa membuat aplikasi pencarian sendiri dengan memanfaatkan fasilitas ini.

Sebagai contoh, di dalam situs **duniaikom**, kotak input SEARCH di kanan atas halaman ini dibuat dengan **method=get**. Misalkan kita ingin mencari seluruh postingan yang berkaitan dengan '**Form PHP**', maka kita bisa mengetikkan kata '**Form PHP**', kemudian men-klik tombol **enter** di **KEYBOARD**. Beberapa saat kemudian, hasil pencarian akan ditampilkan. Jika diperhatikan alamat URL pada web browser, akan berubah menjadi:

<https://www.duniaikom.com/?s=form+php&submit=Search>

Hasil URL ini merupakan query pencarian default yang digunakan oleh **wordpress** (CMS yang digunakan di [duniaikom](#)).

Setelah nama domain **WWW.DUNIAILKOM.COM**, selanjutnya kita melihat karakter **?s=form+php&submit=Search**. Karakter-karakter ini adalah hasil format form dengan **metode=get**. Tanda **?** menandakan awal nilai form, tanda **+** untuk menandakan spasi, dan

tanda & untuk memisahkan objek form. Dengan memanfaatkan keterangan ini, kita bisa melakukan pencarian hanya melalui URL secara langsung, tanpa harus melalui form.

Sebagai contoh, jika anda ingin mencari semua postingan dengan kata kunci 'INTEGER DALAM PHP', maka anda bisa menggunakan link berikut:

<https://www.duniaikom.com/?s=integer+dalam+php&submit=Search>

Untuk mengujinya, silahkan COPY-PASTE text tersebut ke ADDRESS BAR web browser, dan situs **duniaikom** akan langsung menampilkan hasil pencarian dengan kata kunci 'INTEGER DALAM PHP'. Hal ini membuat halaman web menjadi lebih fleksibel.

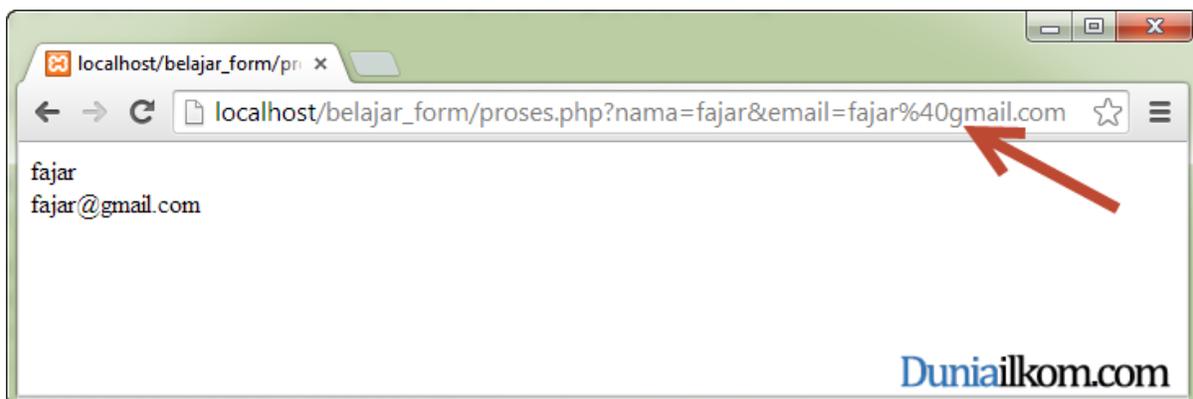
Untuk contoh lainnya, jika anda mengikuti tutorial Form HTML ini dari Part 1, maka kita memiliki file **proses.php** yang berfungsi untuk memproses hasil FORM. Fleksibilitas yang diberikan oleh variabel **\$_GET** membuat kita bisa menampilkan nilai dalam **proses.php** tanpa menggunakan form sama sekali.

Jika isi dari file **proses.php** adalah:

```
<?php
echo $_GET['nama'];
echo "<br />";
echo $_GET['email'];
?>
```

Kita bisa membuat langkah yang sama seperti contoh di dalam pencarian 'INTEGER DALAM PHP' sebelumnya. Sekarang copy paste URL berikut ini kedalam ADDRESS BAR web browser (pastikan aplikasi XAMPP sudah berjalan dan file **proses.php** berada di dalam folder **belajar_form**):

http://localhost/belajar_form/proses.php?nama=fajar&email=fajar%40gmail.com



Dan hasil tampilan halaman **proses.php** akan menampilkan nama FAJAR dan alamat

email FAJAR@GMAIL.COM. Hasil ini didapat hanya dengan menginput nama tersebut dengan URL (tanpa form).

Dari contoh ini kita bisa melihat bahwa PHP mendeteksi nilai dari **method=get**, hanya berdasarkan URL yang diberikan. Cara 'MENGAKALI' variabel **\$_GET** tanpa form ini sering digunakan sebagai sarana untuk mengirim sebuah nilai dari satu halaman PHP ke halaman lainnya (sering digunakan untuk menampilkan pesan error kepada user).

Karena sifat **method=get** yang harus menggunakan **URL** untuk mengirim data form, PHP membatasi hanya bisa memproses 1024 karakter saja. Sehingga jika anda memiliki form yang isinya kemungkinan akan melebihi 1024 karakter, sebaiknya menggunakan **method=post**.

6.2.2 KEUNGGULAN DAN KELEMAHAN METHOD FORM 'POST'

Keuntungan menggunakan **method=post** dalam pembuatan form PHP adalah bahwa isi dari form tidak ditampilkan di **URL**, sehingga method ini sesuai untuk data-data yang bersifat sensitif seperti USERNAME dan PASSWORD.

Sesuai dengan namanya, **method=post** disarankan digunakan untuk form yang digunakan untuk menambah data (**posting data**) atau sebuah proses yang akan mengubah isi database, seperti query INSERT, DELETE. Contoh penggunaan **method=post** ini sering digunakan untuk FORM REGISTER dan FORM LOGIN yang dianggap bersifat rahasia.

Jika kita membuat form untuk keperluan upload file (menggunakan tag **<input type=file>**), maka kita harus membuat form dengan **method=post**.

Sebagai kesimpulan terakhir, pilihan antara **method=get** dengan **method=post** bergantung kepada tingkat SENSITIFITAS data. Untuk data yang sensitif seperti PASSWORD atau yang melakukan suatu perubahan ke database, sebaiknya menggunakan **method=post**. Namun untuk data yang ditujukan untuk menampilkan, atau mengambil sesuatu dari database, sebaiknya menggunakan **method=get**.

6.3 VARIABEL GET POST DAN REQUEST

Variabel **\$_GET** dan **\$_POST** (dan juga **\$_REQUEST**) di dalam PHP termasuk ke dalam kelompok variabel yang dikenal dengan 'Variabel SuperGlobal'.

Variabel SuperGlobals adalah variabel khusus di dalam PHP yang bisa diakses dari halaman PHP manapun tanpa perlu mendefinisikannya terlebih dahulu, dan untuk mengakses variabel ini kita juga tidak perlu menggunakan keyword **global** (sebagaimana variabel global pada umumnya)

Selain variabel `$_GET`, `$_POST` dan `$_REQUEST`, PHP masih memiliki beberapa variabel SUPERGLOBAL lainnya seperti `$_COOKIE`, `$_SESSION`, dan `$_SERVER`. Ciri khusus untuk variabel global di dalam PHP, diawali dengan tanda `$_`. Namun pada tutorial ini kita hanya fokus kepada variabel `$_GET`, `$_POST` dan `$_REQUEST`. Variabel `$_GET`, `$_POST` dan `$_REQUEST` merupakan **tipe data array**, sehingga untuk mengakses nilainya, kita menggunakan cara akses array yakni dengan menggunakan kurung siku seperti: `$_GET['nama']` dimana **nama** adalah nilai dari atribut **name** pada objek form yang akan diakses.

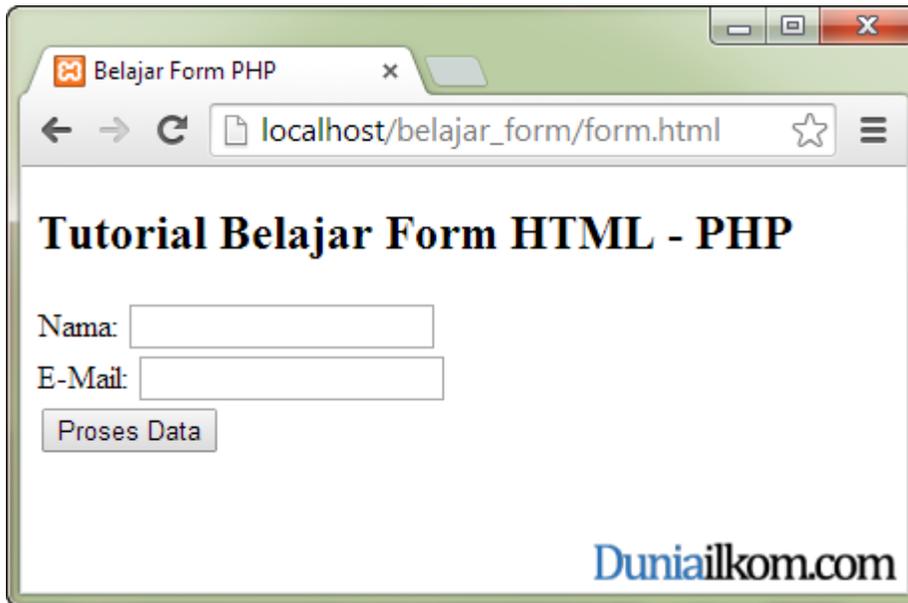
6.3.1 PERBEDAAN VARIABEL GLOBAL `$_GET`, `$_POST` DAN `$_REQUEST`

Seperti yang telah kita praktekan dalam tutorial Cara Menampilkan Hasil Form HTML dengan PHP, kita telah mengetahui bahwa jika form dikirim menggunakan **method=get** maka di dalam PHP kita mengaksesnya dengan variabel `$_GET`, namun jika form dibuat menggunakan **method=post**, kita mengaksesnya dengan variabel `$_POST`.

Bagaimana jika pada saat memproses form kita tidak mengetahui dengan pasti apakah form dikirim dengan **GET** atau **POST**? PHP menyediakan variabel `$_REQUEST` sebagai salah satu solusinya. Variabel `$_REQUEST` menampung nilai form yang dikirim dengan **method=get**, maupun **method=post** secara bersamaan.

Untuk mencobanya, silahkan jalankan file **form.html** dengan isi kode HTML sebagai berikut:

```
<!DOCTYPE html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Belajar Form PHP</title>
</head>
<body>
<h2>Tutorial Belajar Form HTML - PHP </h2>
<form action="proses.php" method="get">
Nama: <input type="text" name="nama" />
<br />
E-Mail: <input type="text" name="email" />
<br />
<input type="submit" value="Proses Data" >
</form>
</body>
</html>
```



Halaman `form.html` diatas persis sama dengan yang kita gunakan pada tutorial sebelumnya, namun untuk halaman `proses.php`, kita akan modifikasi dengan menggunakan variabel `$_REQUEST`:

```
<?php
echo $_REQUEST['nama'];
echo "<br />";
echo $_REQUEST['email'];
?>
```

Jika anda menjalankan `form.html` dan men-klik tombol '**Proses Data**', maka hasil form akan ditampilkan sebagaimana mestinya. Anda juga bisa mengubah method form menjadi `post`, dan variabel `$_REQUEST` akan tetap menampilkan hasil form.

Jadi, variabel apa yang sebaiknya digunakan? apakah `$_GET`, `$_POST` atau `$_REQUEST`? Jawabannya tergantung kepada desain kode program yang dirancang. Jika anda dapat memastikan bahwa form akan dikirim dengan `method=get`, maka gunakan variabel `$_GET`, jika from anda menggunakan `method=post`, maka gunakan `$_POST`, namun jika metodenya tidak dapat dipastikan, variabel superglobal `$_REQUEST` bisa menjadi solusi.

Selain menampung hasil form `get` dan `post`, variabel `$_REQUEST` juga menampung nilai dari `COOKIE`, atau variabel SUPERGLOBALS `$_COOKIE`. Kita akan membahas tentang `COOKIE` dalam tutorial PHP lainnya.

6.4 REGISTER GLOBAL DAN REGISTER LONG ARRAY

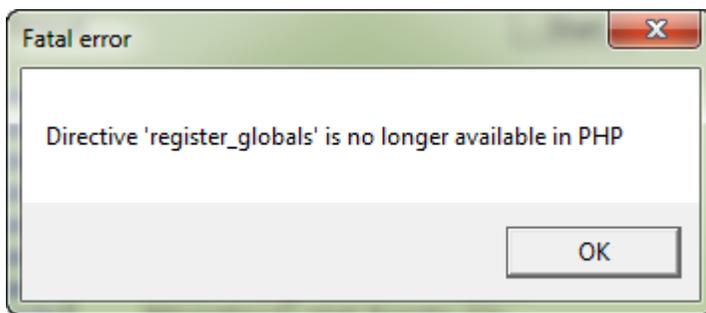
Register Global dan **Register Long Array** adalah metode lain dalam **PHP** yang bisa digunakan untuk menampilkan hasil form, namun kedua metode ini sekarang sudah menyandang status '**deprecated**' yang berarti tidak disarankan digunakan dan kemungkinan tidak akan didukung

oleh **PHP** pada masa mendatang. Kedua metode ini adalah metode 'LEGACY' atau warisan yang digunakan pada versi **PHP** terdahulu.

Saya membahas **Register Global** dan **Register Long Array** hanya sebagai menambah pengetahuan, karena mungkin saja kita masih mendapati kode PHP lama (terutama versi PHP 4 kebawah) yang masih menggunakan kedua metode ini.

Baik **Register Global** dan **Register Long Array** sudah tidak disarankan untuk digunakan karena memiliki celah keamanan yang mudah dibobol. Anda disarankan untuk menggunakan variabel **\$_GET**, **\$_POST** atau **\$_REQUEST**.

Update: **Register Global** dan **Register Long Array** sudah tidak didukung lagi dalam PHP. Jika anda menemukan pesan error pada saat mengaktifkan file PHP.INI, maka anda tidak bisa menjalankan contoh dalam tutorial ini, namun tidak perlu khawatir karena baik **Register Global** dan **Register Long Array** memang tidak disarankan digunakan lagi.



6.4.1 CARA MENGAKSES NILAI FORM DENGAN REGISTER GLOBAL

Register Global adalah sebuah metode dalam PHP yang 'MEMUDAHKAN' pengambilan nilai form dengan cara menjadikan atribut nama dari objek form secara otomatis menjadi variabel di dalam PHP.

Misalkan kita memiliki objek form sebuah inputan dengan kode HTML sebagai berikut:

```
<input type="text" name="user_name" />
```

Maka di dalam halaman **PHP**, akan langsung tersedia sebuah variabel **\$user_name** yang berisi nilai dari objek form tersebut dan siap untuk diakses, terlepas dengan cara apa form tersebut dikirim (apakah **get** atau **post**)

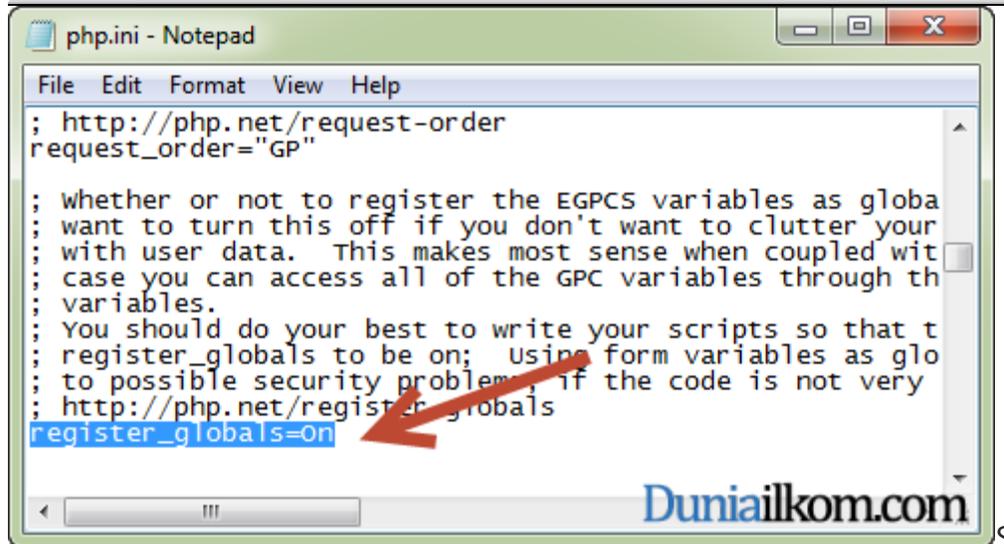
Dalam versi **PHP 5.5** bawaan **XAMPP** yang saya gunakan, **Register Global** sudah tidak didukung lagi. Namun jika anda menggunakan PHP versi lama dan ingin mencobanya, maka kita harus mengubah sebuah settingan PHP dalam file **php.ini**. (cara mengubah setingan file **php.ini** pernah dibahas dalam [Cara Merubah File Konfigurasi php.ini](#))

Bukalah file **php.ini**, lalu cari temukan baris setingan berikut:

```
register_globals=off
```

Kemudian aktifkan fasilitas ini menjadi:

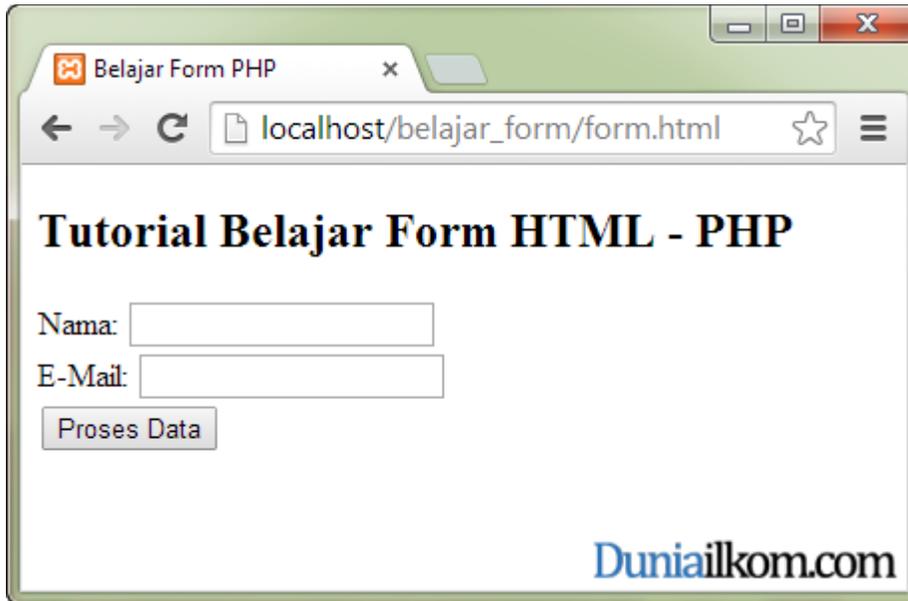
```
register_globals=on
```



Setelah diubah, save file **php.ini**, dan matikan web server apache (dari **XAMPP Control panel**), lalu aktifkan lagi (RESTART). Hal ini diperlukan agar web server Apache dapat membaca perubahan setingan PHP yang baru saja kita lakukan.

Selanjutnya, mari kita gunakan kembali halaman **form.html** sebagai contoh form. Jika anda belum membuah file tersebut, berikut adalah kode HTML yang dibutuhkan:

```
<!DOCTYPE html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Belajar Form PHP</title>
</head>
<body>
<h2>Tutorial Belajar Form HTML - PHP </h2>
<form action="proses.php" method="get">
Nama: <input type="text" name="nama" />
<br />
E-Mail: <input type="text" name="email" />
<br />
<input type="submit" value="Proses Data" />
</form>
</body>
</html>
```



Dalam percobaan **Register Global** ini, kita tidak akan melakukan perubahan apa-apa pada halaman **form.html**, namun pada file **proses.php**, ubahlah menjadi:

```
<?php
echo $nama;
echo "<br />";
echo $email;
?>
```

Silahkan mencoba form tersebut, dan jika setingan **register_globals** telah aktif, maka hasil form akan tampil sebagaimana contoh-contoh kita sebelumnya. Anda juga bisa mengubah metode pengiriman form dari **method=get** menjadi **method=post** dan halaman **proses.php** akan menampilkan hasilnya dengan sukses.

Sepintas cara pengaksesan nilai form seperti ini sangat memudahkan penulisan program, **namun register global** memiliki celah keamanan yang bisa berdampak fatal. Hal ini terjadi karena pengunjung website bisa 'MEMASUKKAN' variabel PHP ke dalam halaman kita.

Misalkan kita memiliki sebuah halaman **login.php** yang berfungsi untuk memproses hak akses untuk bisa masuk ke situs, dan kode PHP nya adalah sebagai berikut:

```
<?php
// cek apakah user ada di database
mysql_connect("localhost","root","");
mysql_select_db("mahasiswa");
$user_db="SELECT * FROM tabel_user WHERE user='$user_name'";
$match = mysql_num_rows($user);

//jika user valid, ubah variabel $hak_akses menjadi true
if ($match)
{
```

```
$hak_akses=true;
}

//jika hak akses true, masuk ke proses utama
if ($hak_akses==true)
{
// kode program jika user valid
}
?>
```

Sekarang, jika **register_global = on**, seseorang bisa 'MELOMPATI' bagian validasi tersebut dengan menuliskan alamat URL berikut:

```
login.php?hak_akses=true
```

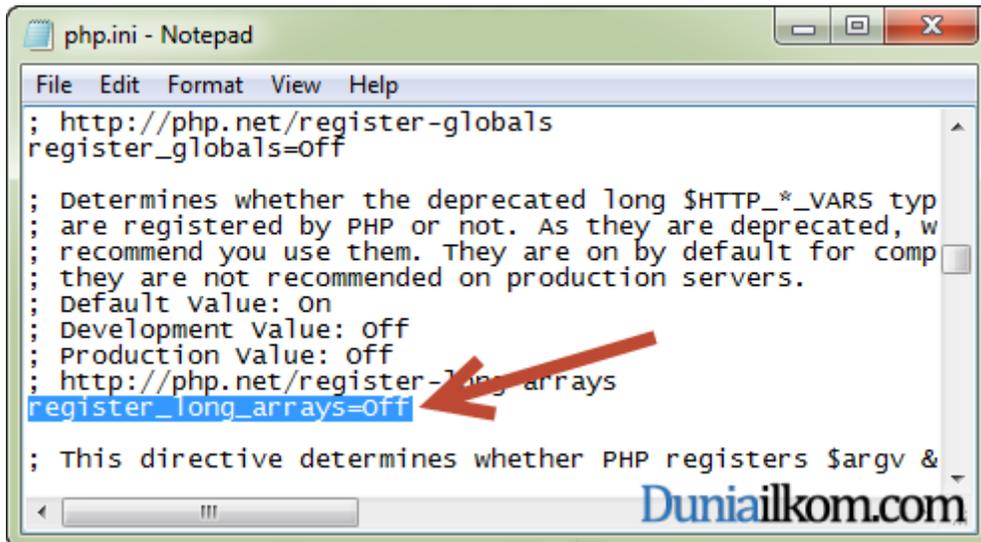
Dengan mengetikkan alamat tersebut, variabel **\$hak_akses** akan langsung bernilai **true** di dalam halaman **login.php**, sehingga seseorang bisa mengakses sistem kita tanpa melalui proses VALIDASIuser. Karena hal inilah, mengaktifkan **register global** dianggap berbahaya dan tidak disarankan.

6.4.2 CARA MENGAkses NILAI FORM DENGAN REGISTER LONG ARRAY

Cara lain untuk mengambil nilai dari form adalah menggunakan **Register Long Array**. Sesuai dengan namanya, **Register Long Array** memiliki nama variabel yang lebih panjang jika dibandingkan variabel biasa (variabel **\$_GET** dan **\$_POST**).

Untuk mengambil nilai form yang dikirim dengan **method=get**, kita menggunakan **\$HTTP_GET_VARS**, dan untuk **method=post** menggunakan **\$HTTP_POST_VARS**.

Sama seperti **Register Globals**, **Register Long Array** tidak didukung lagi untuk PHP versi terbaru. Jika anda memiliki PHP versi lama dan ingin mencobanya, lakukan perubahan settingan **php.ini** untuk pilihan **register_long_arrays=on**, seperti gambar berikut ini:



Setelah itu, restart WEB SERVER **apache** menggunakan **XAMPP Control Panel**.

Sebagai contoh, masih menggunakan halaman **form.html**, kali ini saya akan mengubah halaman **proses.php** menjadi:

```
<?php
echo $HTTP_GET_VARS['nama'];
echo "<br />";
echo $HTTP_GET_VARS['email'];
?>
```

Dan jika tidak ada masalah, kode program tersebut akan menampilkan hasil inputan form dari halaman **form.html**. **Register Long Array** tidak disarankan lagi karena **PHP** telah memperkenalkan metoda variabel **\$_GET** dan **\$_POST** yang lebih praktis. Dalam tutorial kali ini, kita telah mempelajari cara mengakses nilai form menggunakan [Register Globals dan Register Long Array](#). Kedua cara ini sudah tidak digunakan lagi, dan saya membahasnya karena mungkin kita masih menemukan kode PHP lama yang masih menggunakannya.

6.5 VALIDASI FORM PHP

Setelah berhasil mengambil dan menampilkan nilai dari form, hal berikutnya yang harus kita lakukan terhadap data tersebut adalah melakukan proses **validasi**. Proses validasi dilakukan terhadap nilai yang dimasukkan melalui **form**.

Dalam tutorial kali ini kita akan membahas [Cara Membuat Validasi Form dengan PHP](#) menggunakan fungsi **isset()** dan fungsi **empty()**.

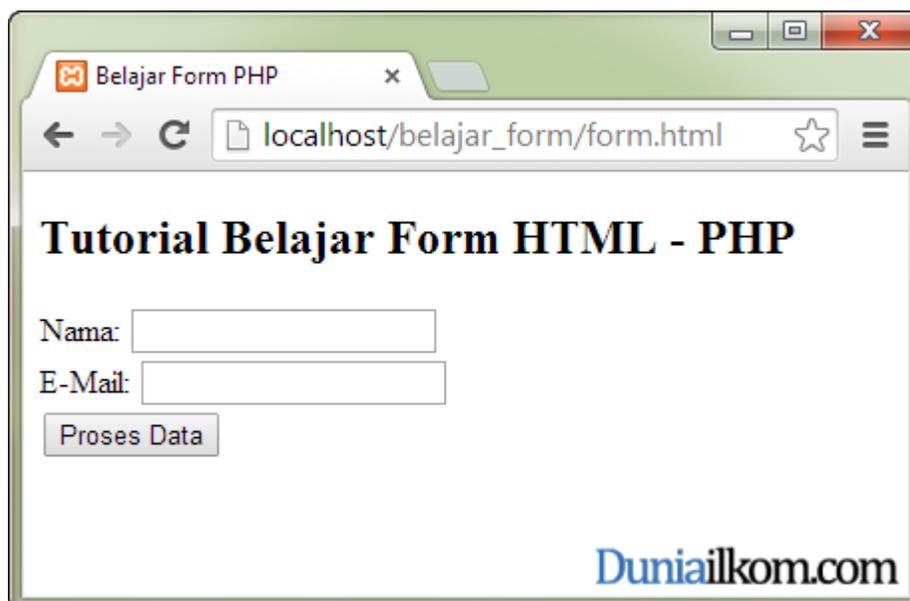
6.5.1 PENTINGNYA MELAKUKAN VALIDASI NILAI FORM

Nilai yang telah diinput oleh user atau pengunjung web, tidak bisa begitu saja di simpan langsung ke dalam database. Karena kita tidak tahu apakah nilai tersebut telah sesuai dengan nilai yang kita kehendaki. Misalkan apakah nilai tersebut harus berupa angka, atau hanya bisa berupa huruf, atau apakah hanya bisa diinput dalam range tertentu saja.

Dalam kasus yang EKSTRIM, seorang user bisa saja memasukkan KODE SCRIPT atau tag HTML yang bisa merusak situs kita, hal ini dikenal dengan **Cross-site Scripting**. Sebuah proses **validasi** nilai merupakan hal yang sangat penting dalam merancang form. Khusus untuk validasi mencegah **Cross-site Scripting** dan juga **HTML injection** ini akan saya bahas pada tutorial form PHP berikutnya.

Dalam pembahasan tutorial validasi form ini, saya masih menggunakan contoh halaman **form.html** yang pernah kita buat pada tutorial sebelumnya, berikut adalah kode HTML untuk halaman form.html:

```
<!DOCTYPE html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Belajar Form PHP</title>
</head>
<body>
<h2>Tutorial Belajar Form HTML - PHP </h2>
<form action="proses.php" method="get">
Nama: <input type="text" name="nama" />
<br />
E-Mail: <input type="text" name="email" />
<br />
<input type="submit" value="Proses Data" />
</form>
</body>
</html>
```

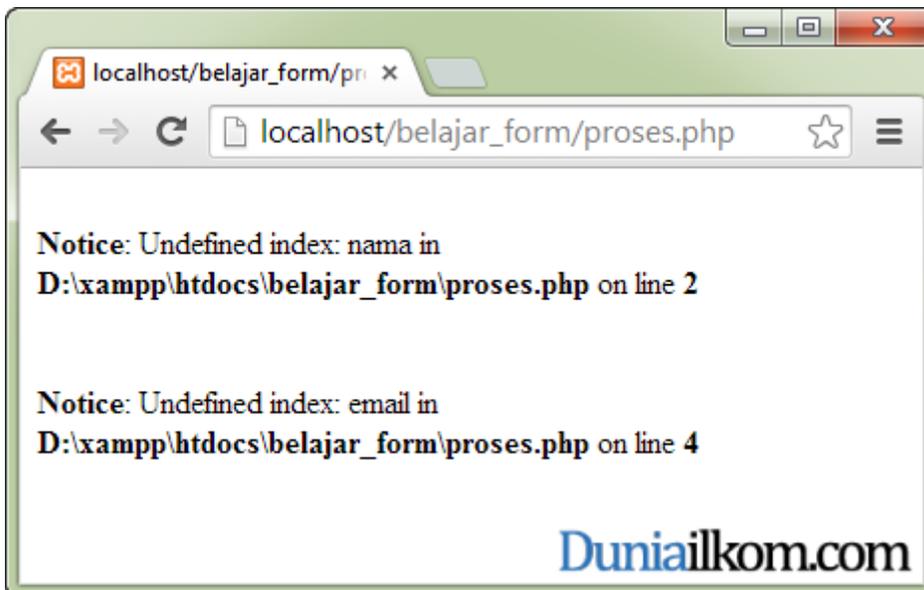


6.5.2 MEMERIKSA KETERSEDIAAN VARIABEL FORM DENGAN FUNGSI ISSET()

Validasi pertama yang paling sederhana dan 'HAMPIR' selalu ada dalam tiap proses validasi form dalam PHP adalah memeriksa apakah objek form tersebut sudah tersedia atau tidak. Sebagai contoh sederhananya: apakah variabel `$_GET['nama']` tersedia untuk diproses atau tidak.

Proses memeriksa 'KETERSEDIAAN' variabel ini menjadi penting karena **PHP** akan mengeluarkan pesan peringatan jika kita mengakses nilai sebuah variabel yang belum didefinisikan terlebih dahulu.

Sebagai contoh, jika kita mengakses langsung halaman **proses.php** (tanpa melalui halaman **form.html**) dan tanpa menambahkan URL (seperti pembahasan pada tutorial [Perbedaan Metode Pengiriman Form GET dan POST](#)) PHP akan menampilkan pesan peringatan seperti berikut ini:



Notice: Undefined index adalah pesan error yang terjadi karena kita langsung menampilkan variabel `$_GET['nama']` dan `$_GET['email']` yang memang belum diset sebelumnya.

Untuk memeriksa apakah sebuah objek form telah didefinisikan atau telah di-set sebelumnya, kita bisa menggunakan fungsi bawaan PHP: `isset()`. Fungsi `isset()` akan menghasilkan nilai **true** jika sebuah variabel telah didefinisikan, dan **false** jika variabel tersebut belum dibuat.

Sebagai langkah antisipasi, saya akan membuat proses **validasi** untuk menangani variabel `$_GET` yang belum di-set, berikut adalah modifikasi file **proses.php**:

```
<?php
if (isset($_GET['nama']))
{
echo $_GET['nama'];
}
```

```

echo "<br />";

if (isset($_GET['email']))
{
echo $_GET['email'];
}
?>

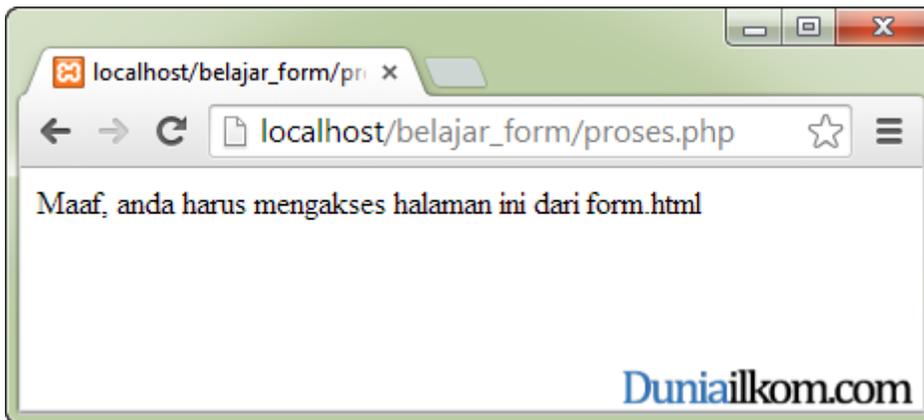
```

Sekarang, file **proses.php** tidak akan menghasilkan error apabila diakses tanpa melalui **form.html**. Namun perubahan kode tersebut tidak terlalu berguna karena tidak memberikan pesan error yang jelas. Berikut adalah modifikasi file **proses.php** agar lebih informatif:

```

<?php
if (isset($_GET['nama']) AND isset($_GET['email']))
{
echo $_GET['nama'];
echo $_GET['email'];
}
else
{
echo "Maaf, anda harus mengakses halaman ini dari form.html";
}
?>

```



Pada kode PHP diatas saya mengharuskan nilai `$_GET['nama']` dan `$_GET['email']` tersedia, baru nilai ditampilkan, namun jika tidak ada, akan ditampilkan pesan bahwa halaman ini hanya bisa diakses dari **form.html**.

6.5.3 MEMERIKSA APAKAH VARIABEL FORM TELAH DIISI

Fungsi `isset()` yang kita bahas sebelumnya hanya memeriksa apakah sebuah objek form ada atau tidak. Fungsi `isset()` tetap bernilai **true** meskipun user tidak mengisi form sama sekali (variabel form bernilai kosong, namun variabel tersebut dianggap telah di-set).

Untuk memeriksa apakah sebuah objek form telah diisi atau tidak, kita bisa menggunakan fungsi: `empty()`.

Fungsi **empty()** akan menghasilkan nilai **false** jika sebuah variabel telah diisi, dan bernilai **true** jika variabel tersebut belum diisi. Dengan menggunakan stuktur **IF** dan fungsi **empty()**, kita bisa membuat logika validasi objek form mana saja yang dianggap perlu (harus diisi) dan mana yang boleh dikosongkan. Dan kemudian menampilkan pesan error yang sesuai.

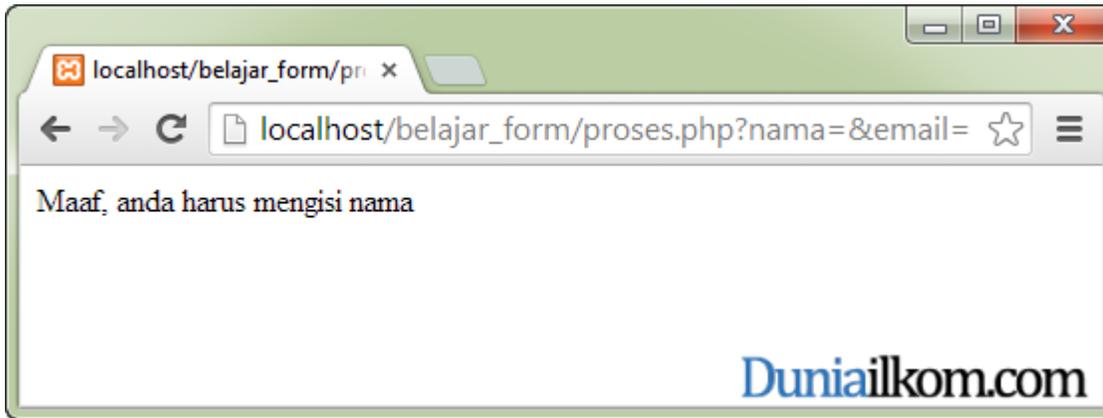
Sebagai contoh, saya akan memodifikasi file **proses.php** agar menampilkan pesan error jika kotak input **nama** tidak diisi. Berikut adalah kode PHP pada halaman proses.php:

```
<?php
if (isset($_GET['nama']) AND isset($_GET['email']))
{
$nama=$_GET['nama'];
$email=$_GET['email'];
}
else
{
die("Maaf, anda harus mengakses halaman ini dari form.html");
}

if (!empty($nama))
{
echo "Nama: $nama <br /> Email: $email";
}
else
{
die("Maaf, anda harus mengisi nama");
}
?>
```

Dalam kode PHP diatas, saya memodifikasi beberapa bagian kode program. Pada **logika IF** pertama, saya melakukan pengecekan apakah variabel **\$_GET['nama']** dan **\$_GET['email']** tersedia atau tidak. Jika tersedia maka pindahkan nilainya ke variabel **\$nama** dan **\$email** agar lebih mudah untuk diproses. Namun jika tidak, fungsi **die()** akan menghentikan proses dan menampilkan pesan kesalahan.

Pada logika IF kedua, saya memeriksa apakah variabel **\$nama** kosong atau tidak dengan fungsi **!empty()**. Fungsi **!empty(\$nama)** akan menghasilkan nilai **true** hanya jika variabel **\$nama** tidak kosong (perhatikan tanda **!** sebagai pembalik logika **empty()**). Namun jika **\$nama** ternyata kosong (tidak diisi), maka tampilkan pesan kesalahan.



6.5.4 MENYELEKSI TIPE DATA OBJEK FORM

Setelah objek form dipastikan tersedia, tidak kosong, validasi berikutnya yang biasanya dilakukan adalah memastikan tipe data dan range data yang diinput oleh user.

Untuk mengecek tipe data sebuah variabel, PHP menyediakan beberapa fungsi tergantung tipe datanya, yakni fungsi `is_string()`, `is_int()`, `is_float()`, `is_numeric()`, `is_bool()`, `is_array()`, dan `is_object()`. Sesuai dengan namanya, masing-masing fungsi tersebut akan mengecek tipe data dari variabel yang ditest.

Diantara fungsi-fungsi diatas, fungsi `is_numeric()` mungkin butuh sedikit penjelasan. Fungsi `is_numeric()` akan mengecek apakah sebuah tipe data merupakan angka baik itu **float** atau **integer**.

Khusus objek form variabel angka seperti umur, biasanya selain menyeleksi apakah nilainya berupa angka **integer**, kita mungkin juga menambahkan aturan bahwa nilai umur harus diatas 17 tahun. Untuk menambahkan fungsi ini, fungsi `is_int()` dapat dikombinasikan dengan **struktur IF**.

Sebagai contoh saya akan menambahkan validasi untuk **tag input** NAMA bahwa nama tidak boleh diisi dengan angka. Untuk keperluan ini saya akan menggunakan fungsi `is_numeric()`. Berikut adalah modifikasi file `proses.php`:

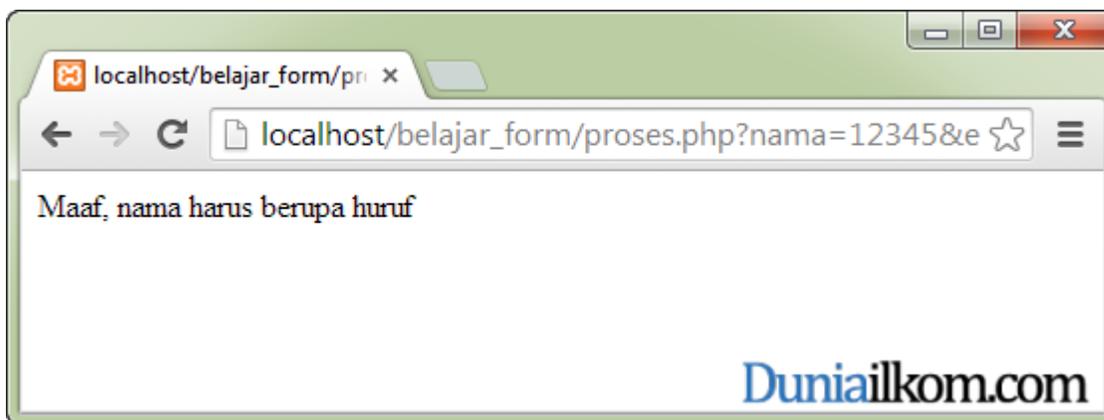
```
<?php
if (isset($_GET['nama']) AND isset($_GET['email']))
{
    $nama=$_GET['nama'];
    $email=$_GET['email'];
}
else
{
    die("Maaf, anda harus mengakses halaman ini dari form.html");
}
```

```

if(empty($nama))
{
die("Maaf, anda harus mengisi nama");
}
else
{
if (is_numeric($nama))
{
die("Maaf, nama harus berupa huruf");
}
else
{
echo "Nama: $nama <br /> Email: $email";
}
}
?>

```

Dalam kode diatas, saya menambahkan 1 lagi **logika IF** untuk menyeleksi apakah variabel **\$nama** berisi angka numerik (**INTEGER** atau **FLOAT**). Jika **\$nama** bertipe numerik, maka tampilkan pesan error.



Di dalam tutorial form PHP kali ini kita telah mempelajari [cara memvalidasi nilai inputan form](#). Namun apa yang telah kita pelajari disini hanya sebagian kecil dari proses validasi yang sebenarnya harus dilakukan. Misalnya, untuk memfilter variabel **\$nama** diatas, akan lebih cocok menggunakan REGULAR EXPRESSION daripada fungsi **is_numeric()**, namun yang penting kita telah bisa 'MENANGKAP' cara pembuatan validasi form.

Di dalam tutorial berikutnya, masih berkaitan dengan proses validasi form, kita akan mempelajari tentang Cara Mencegah Cross-site Scripting dan HTML injection.

6.6 MENCEGAH XSS DAN HTML INJECTION

Cross-site Scripting atau sering disingkat dengan **XSS** adalah jenis serangan ke sebuah situs dengan cara 'MENYISIPKAN' **kode script** (biasanya **JavaScript**) ke dalam sebuah situs. Hal ini hanya akan berhasil jika situs tersebut memiliki fitur untuk menampilkan kembali isian form ke web browser, seperti

form komentar. Sedangkan **HTML injection** adalah istilah yang lebih spesifik kepada cara 'MENYISIPKAN' kode **HTML** kedalam sebuah situs.

Sebagai programmer web, penanganan untuk **Cross-site Scripting** maupun **HTML injection** merupakan hal yang sangat penting, terutama dalam pembuatan kode form dengan **PHP**. Karena form pada dasarnya dapat diinput oleh siapa saja, maka kita perlu memproteksi situs dari kode-kode berbahaya yang bisa diinput oleh user melalui form.

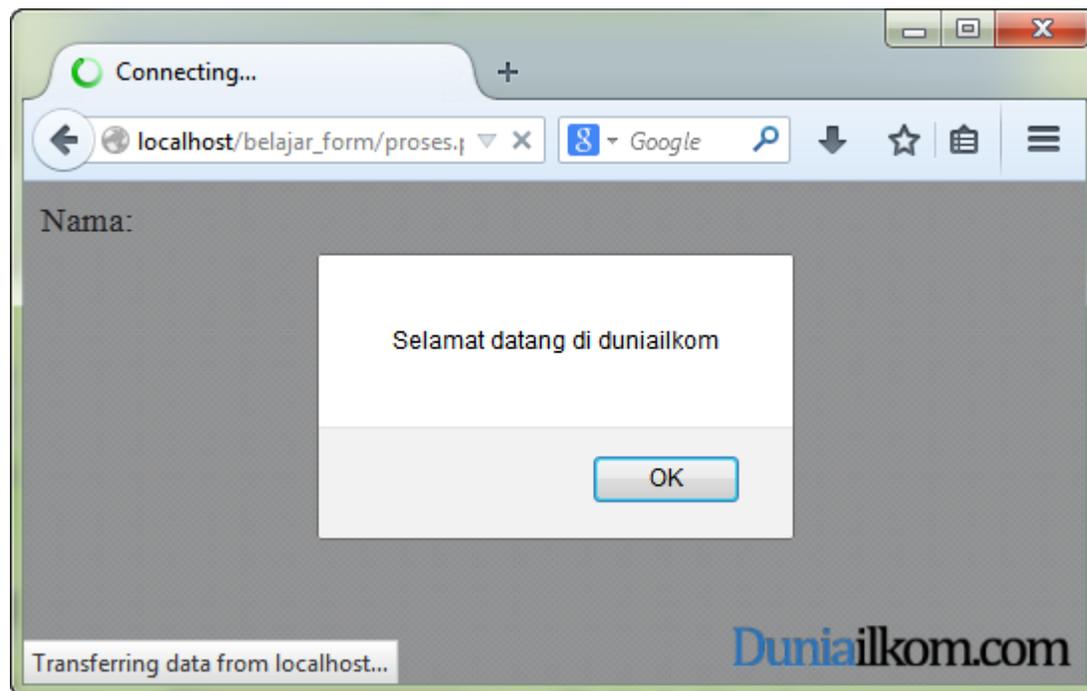
6.6.1 CONTOH CROSS-SITE SCRIPTING DAN HTML INJECTION

Agar lebih memahami maksud dari **Cross-site Scripting** dan **HTML injection**, kita akan coba menginputnya melalui contoh halaman **form.html** dan **proses.php** dari tutorial sebelumnya.

Silahkan buka halaman form kita (**form.html**), lalu input kode dibawah ini kedalam dalam salah satu kotak inputan form:

```
<script>alert('Selamat datang di duniaikom')</script>
```

Dan anda akan mendapati tampilan berikut:



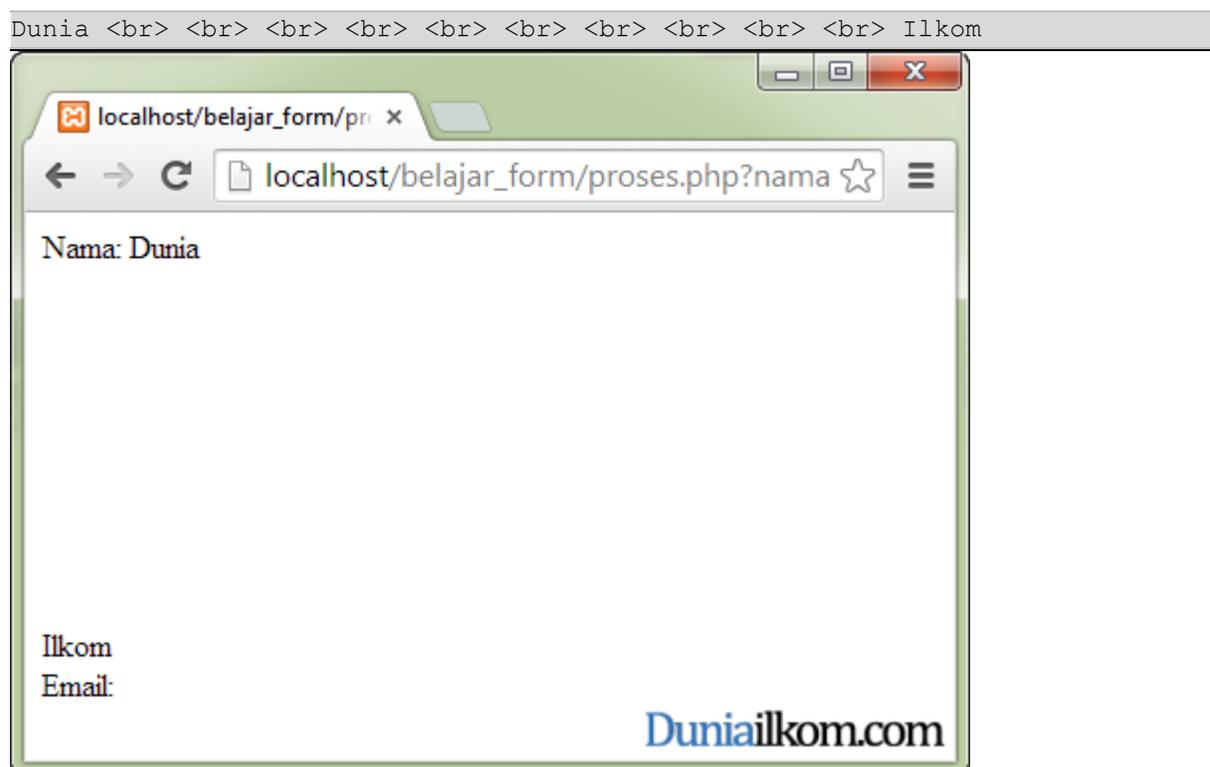
Tampilan

tersebut adalah hasil dari kode **JavaScript** yang baru saja kita input melalui form. Kode tersebut dapat berjalan karena pada halaman **proses.php** kita langsung menampilkan data yang diinput oleh user tanpa melakukan proses filter.

Hal ini sangat berbahaya karena dengan kode **JavaScript** seseorang bisa melakukan 'HAMPIR SEGALANYA' dengan situs kita. Tidak hanya sekedar menampilkan ucapan selamat seperti kode diatas, tetapi seseorang juga bisa merubah BACKGROUND, mengubah tampilan seluruh web, bahkan mengarahkan pengunjung ke situs lain (REDIRECT).

Hasil diatas saya peroleh menggunakan web browser **mozilla firefox**. Ketika saya menggunakan **google chrome**, tampilan javascript tersebut tidak muncul karena google chrome memiliki mekanisme 'PENEGAHAN' kode javascript internal ketika diinput melalui form.

Sebagai contoh lainnya, seseorang bisa menginput kode berikut ke dalam kotak input nama:



Walaupun kode diatas tidak mengandung SCRIPT, namun kode **HTML** tersebut akan membuat tampilan hasil form menjadi berantakan (bayangkan jika kita memiliki halaman yang berisi tabel berisi nama-nama seluruh USER, dan kode diatas akan menghancurkan desain web yang telah dirancang).

6.6.2 CARA MENCEGAH CROSS-SITE SCRIPTING DAN HTML INJECTION

Salah satu cara sederhana untuk menghindari **Cross-site Scripting** dan **HTML injection** adalah dengan membuat karakter-karakter yang memiliki 'MAKNA' di dalam HTML dan JavaScript untuk diubah menjadi **named entity**, yaitu menkonversi karakter khusus seperti < menjadi **<**, dan karakter > menjadi **>**. Atau cara lainnya adalah dengan menghilangkan sama sekali seluruh **tag HTML** atau **SCRIPT** dari inputan user.

Penjelasan lebih lanjut tentang named entity (dan juga numeric entity) karakter HTML pernah kita bahas pada tutorial [Cara Memasukkan Karakter Khusus ke dalam HTML](#).

Untuk kedua keperluan ini, PHP memiliki fungsi `htmlspecialchars()` dan fungsi `strip_tags()`.

Fungsi `htmlspecialchars()` akan mengkonversi 4 karakter 'KHUSUS' HTML menjadi **named entity** sehingga tidak akan di 'PROSES' oleh web browser. Keempat karakter tersebut adalah: <, >, & dan ". Keempat karakter khusus inilah yang membuat web browser akan menerjemahkan sebuah string menjadi kode HTML/JavaScript.

Sedangkan fungsi `strip_tags()` akan menghapus seluruh **tag HTML** dari inputan user.

Sebagai contoh, kita akan mengupdate halaman `proses.php` agar bisa mencegah seseorang menyisipkan **Cross-site Scripting** dan **HTML injection**. Berikut adalah perintah PHP pada file `proses.php`:

```
<?php
if (isset($_GET['nama']) AND isset($_GET['email']))
{
    $nama=$_GET['nama'];
    $email=$_GET['email'];
    $nama=htmlspecialchars($nama);
    $email=strip_tags($email);
}
else
{
    die("Maaf, anda harus mengakses halaman ini dari form.html");
}

if(empty($nama))
{
    die("Maaf, anda harus mengisi nama");
}
else
{
    if (is_numeric($nama))
    {
        die("Maaf, nama harus berupa huruf");
    }
    else
    {
        echo "Nama: $nama <br /> Email: $email";
    }
}
?>
```

Di dalam kode diatas, saya hanya menambahkan fungsi `htmlspecialchars()` untuk variabel `$nama`, dan fungsi `strip_tags()` untuk variabel `$email`. Untuk mengujinya, silahkan anda coba memasukkan kode JavaScript kita sebelumnya ke dalam kotak input nama, yakni:

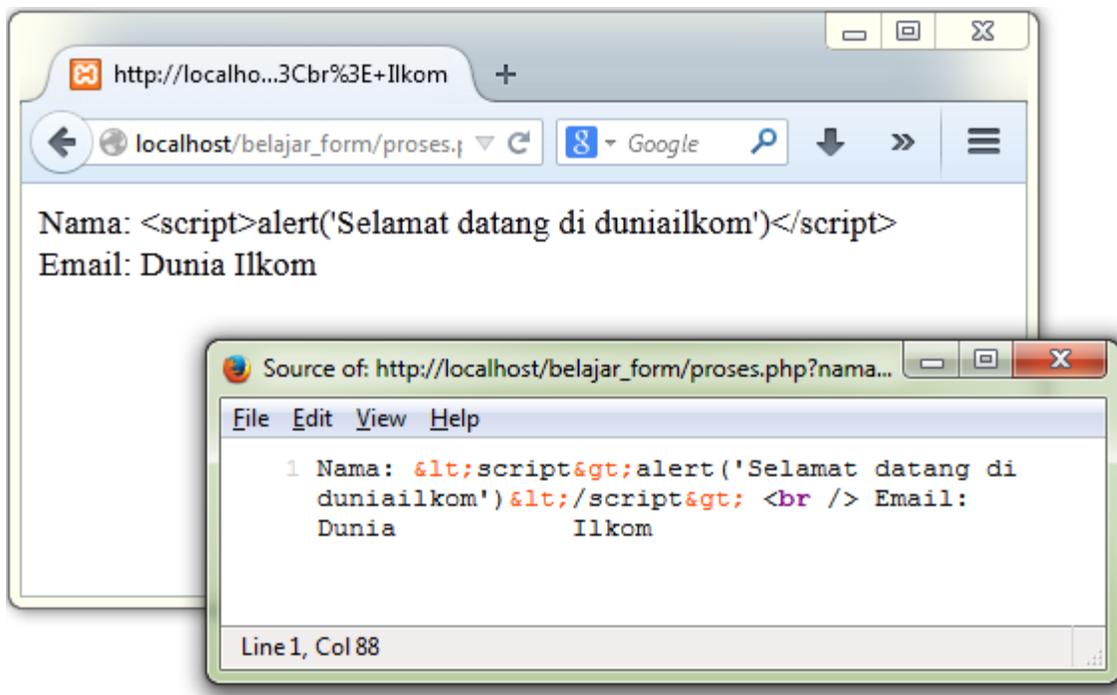
```
<script> alert('Selamat datang di duniailkom')</script>
```

dan nilai berikut ke dalam kotak input email:

```
Dunia <br> Ilkom
```



Kemudian perhatikan hasil tampilan dari proses.php. Seperti yang dirancang, kali ini hasil inputan tersebut akan difilter agar lebih aman.



Jika anda melihat kode yang dihasilkan, maka akan terlihat bahwa fungsi `htmlspecialchars()` akan mengubah

```
<script> alert('Selamat datang di duniailkom')</script>
```

menjadi

```
&lt;script&gt; alert('Selamat datang di duniailkom')&lt;/script&gt;
```

Perhatikan bahwa karakter khusus seperti < dan > telah diubah menjadi **named entity**. Sedangkan untuk kotak inputan email, seluruh tag
 telah di filter dan dihapus secara otomatis.

6.6.3 VALIDASI DENGAN HTMLSPECIALCHARS ATAU STRIP_TAGS?

Pilihan apakah menggunakan fungsi **htmlspecialchars()** atau **strip_tags()** tergantung kepada alur logika form yang kita buat. Untuk isian form seperti **user_name**, **email**, dan **alamat** akan lebih aman jika kita tidak membolehkan tag HTML sama sekali (menggunakan fungsi **strip_tags()**), namun untuk kotak form komentar, mungkin kita akan membolehkan beberapa tag untuk diproses seperti tag , tag <i>, atau mungkin juga tag <a>.

Fungsi **strip_tags()** memiliki argumen kedua yang bisa diisi dengan tag-tag HTML apa saja yang 'dibolehkan'. Jika kita membolehkan tag <a>, dan tag <i> untuk variabel **\$komentar**, maka penulisannya bisa dibuat menjadi:

```
$komentar=strip_tags($komentar, '<a><b><i>').
```

Sedangkan untuk fungsi **htmlspecialchars()**, jika anda ingin karakter tanda kutip (') juga diubah menjadi **named entity**, kita bisa menambahkan 1 argumen optional untuk fungsi **htmlspecialchars()**, dengan pilihan **ENT_QUOTES**. Sehingga dalam contoh variabel \$nama, kita bisa menulisnya menjadi:

```
$nama=htmlspecialchars($nama, ENT_QUOTES);
```

Dengan demikian, jika diinput:

```
<script>alert('Selamat datang di duniailkom')</script>
```

Akan diproses menjadi:

```
&lt;script&gt; alert(&#039;Selamat datang di duniailkom&#039;)&lt;/script&gt;
```

Perhatikan bahwa karakter kutip (') diganti menjadi kode **'**.

6.6.4 MENGENAL FUNGSI HTMLENTITIES()

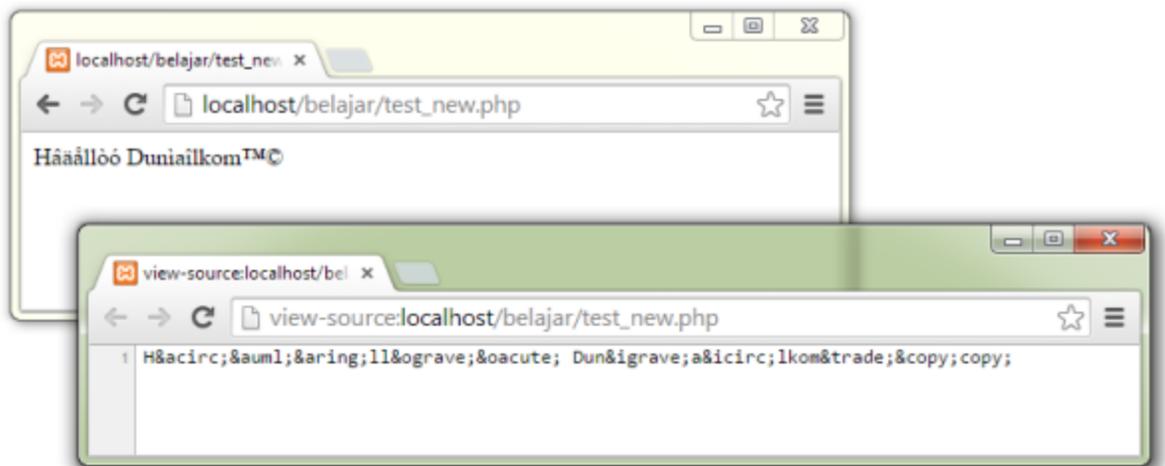
Selain fungsi **htmlspecialchars()** dan fungsi **strip_tags()**, dalam pembahasan mengenai validasi dan karakter entity HTML, PHP masih memiliki 1 lagi fungsi yang sering dipakai, yakni fungsi **htmlentities()**. Fungsi **htmlentities()** akan mengkonversi seluruh karakter khusus di dalam sebuah string menjadi

entity, tidak hanya karakter <, >, & dan " seperti pada fungsi `htmlspecialchars()`. Fungsi `htmlentities()` akan memproses karakter khusus lain seperti: ™, ©, ®, atau Σ menjadi **named entity**.

Sebagai contoh, berikut adalah hasil konversi fungsi `htmlentities()`:

```
<?php
$string = "Hääällòò Duniaïlkom™©";
echo htmlentities($string);
?>
```

Dan berikut adalah hasilnya:



Duniaïlkom.com

Seperti yang terlihat bahwa fungsi `htmlentities()` akan mengubah seluruh karakter khusus menjadi named entity HTML.

6.7 MENGIRIM VARIABEL ANTAR HALAMAN PHP

Dalam **PHP** terdapat beberapa cara untuk mengirim **VARIABEL** antar halaman. Salah satunya adalah dengan memanfaatkan cara pengiriman form dengan **method=GET**. Jika kita mengirim form dengan metode **GET**, maka hasil dari form akan ditambahkan di dalam menggunakan **VARIABEL SUPERGLOBAL \$_GET**. Sebagai contoh, jika di dalam form terdapat kotak input dengan atribut **name=user_name** yang diisi dengan '**duniaïlkom**', maka ketika dikirim untuk diproses ke halaman **proses.php**, di dalam **URL** akan menjadi:

```
proses.php?user_name=duniaïlkom
```

dan didalam halaman **proses.php**, variabel `$_GET['user_name']` akan berisi '**DUNIAILKOM**'.

Dengan cara ini kita bisa mengirim variabel dari sebuah halaman ke halaman lain. Misalkan kita membuat sebuah kondisi logika dengan PHP, dan jika logikanya salah maka tampilkan sebuah link yang menyatakan bahwa alamat tidak lengkap. Pengiriman pesan ini bisa ditulis sebagai berikut:

```
proses.php?pesan=alamat+tidak+lengkap.
```

Ketika user men-klik link tersebut, pada halaman **proses.php** akan memiliki sebuah variabel `$_GET['pesan']` yang berisi: 'ALAMAT TIDAK LENGKAP'. Kita bisa memanfaatkan ini untuk berbagai hal, dan dalam tutorial kali ini kita akan menggunakan cara ini untuk menampilkan pesan kesalahan dari form. Untuk dapat mengirimkan variabel melalui URL, terdapat beberapa aturan penggunaan karakter. Berikut adalah contoh hasil pengiriman dari sebuah halaman:

```
proses.php?nama=dunia_ilkom&email=duniailkom@gmail.com&pesan=welcome
```

perhatikan bahwa tanda '?' digunakan untuk mengawali pengiriman variabel, tanda '=' digunakan sebagai pengisi nilai variabel, tanda '&' untuk memisah'kan antara variabel yang satu dengan yang lain, dan tanda '+' sebagai pengganti karakter 'SPASI'.

6.7.1 REDIRECT HALAMAN DENGAN FUNGSI HEADER(LOCATION:)

Fungsi `header()` di dalam PHP digunakan untuk memproses **HTTP header**. Fungsi `header()` memiliki banyak fitur, namun yang akan kita manfaatkan adalah fitur untuk mengirim user ke halaman lain atau dikenal dengan istilah: **redirect**.

Untuk 'MENGIRIM' user ke sebuah halaman, kita menggunakan **Location:halaman.html** sebagai argumen fungsi `header()`. Misalkan kita ingin mengirim user ke halaman **pendaftaran.html** pada situs **duniailkom.com**, maka penulisan fungsi `header()` adalah sebagai berikut:

```
header("Location: https://www.duniailkom.com/pendaftaran.html")
```

Halaman yang dituju tidak harus berupa halaman **HTML**, tetapi juga halaman **PHP**, atau bahkan file multimedia seperti gambar. Fitur ini akan kita manfaatkan untuk menampilkan pesan kesalahan form dalam PHP.

6.7.2 CARA MENAMPILKAN PESAN KESALAHAN FORM

Dalam tutorial sebelumnya tentang [Validasi Form untuk Mencegah Cross-site Scripting dan HTML injection](#), kita telah membuat proses validasi form dan menampilkan pesan kesalahan form berdasarkan kondisi yang terjadi. Namun pesan kesalahan tersebut hanya ditampilkan pada halaman **proses.php**. Jika user ingin mengubah isian form, ia terpaksa kembali kehalaman **form.html** untuk memperbaiki kesalahan form tersebut dengan cara mengetik manual di **web browser**. Salah satu fitur pengisian form

yang umum digunakan adalah ketika isian form ditemukan kesalahan, maka user akan dikembalikan ke halaman awal form, beserta pesan kesalahan. Untuk menghasilkan fitur seperti ini, kita akan memanfaatkan fungsi **PHP: header()**.

Kembali menggunakan halaman **form.html** dan **proses.php** yang kita gunakan dalam tutorial-tutorial sebelumnya, kali ini saya akan memodifikasi halaman **proses.php** agar bisa memanfaatkan fungsi **header()**. Berikut adalah modifikasi file **proses.php**:

```
<?php
if (isset($_GET['nama']) AND isset($_GET['email']))
{
    $nama=$_GET['nama'];
    $email=$_GET['email'];
    $nama=htmlspecialchars($nama);
    $email=strip_tags($email);
}
else
{
    header("Location:form.php?error=variabel_belum_diset");
}

if(empty($nama))
{
    header("Location:form.php?error=nama_kosong");
}
else
{
    if (is_numeric($nama))
    {
        header("Location:form.php?error=nama_harus_huruf");
    }
    else
    {
        echo "Nama: $nama <br /> Email: $email";
    }
}
?>
```

Dari kode **proses.php** kita sebelumnya, saya hanya mengubah bagian error menggunakan fungsi **header()**. Namun perhatikan bahwa saya mengirimkan variabel error kepada halaman **form.php**, pesan inilah yang akan diproses pada halaman **form.php**. Untuk halaman **form.html**, saya akan mengubah nama file menjadi **form.php**. Perubahan ini diperlukan karena pada halaman **form.html** memerlukan proses PHP untuk menangani variabel error yang dikirim dari halaman **proses.php**.

Berikut adalah kode dari halaman **form.php**:

```
<?php
//ambil nilai variabel error
if (isset($_GET['error']))
```

```

{
$error=$_GET['error'];
}
else
{
$error="";
}

//siapkan pesan kesalahan
$pesan="";
if ($error=="variabel_belum_diset")
{
$pesan="<h3>Maaf, anda harus mengakses halaman ini dari form.php</h3>";
}
if ($error=="nama_kosong")
{
$pesan="<h3>Maaf, anda harus mengisi nama</h3>";
}
if ($error=="nama_harus_huruf")
{
$pesan="<h3>Maaf, nama harus berupa huruf</h3>";
}
?>

<!DOCTYPE html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Belajar Form PHP</title>
</head>
<body>
<h2>Tutorial Belajar Form HTML - PHP </h2>
<?php
echo $pesan;
?>
<form action="proses.php" method="get">
Nama: <input type="text" name="nama" />
<br />
E-Mail: <input type="text" name="email" />
<br />
<input type="submit" value="Proses Data" />
</form>
</body>
</html>

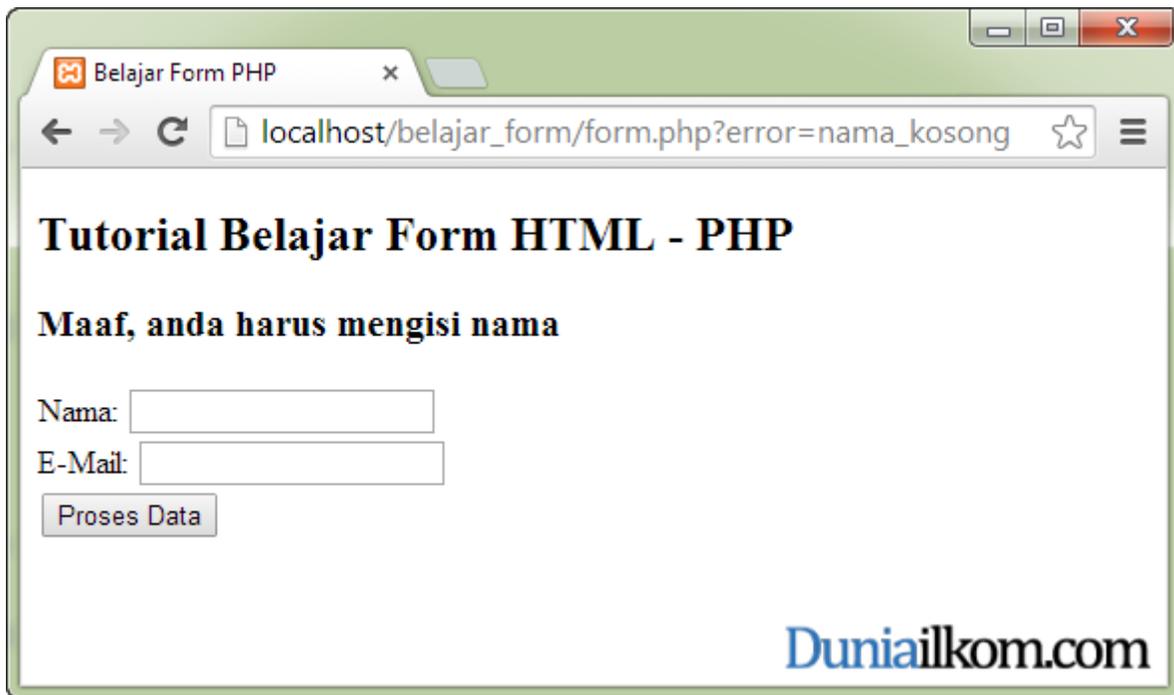
```

Pada halaman **form.php** saya menambahkan beberapa kode PHP di awal halaman. Kode PHP ini berfungsi untuk menangani variabel error yang dikirimkan dari halaman **proses.php**.

Setelah memindahkan variabel **\$_GET['error']** ke dalam variabel **\$error** agar mudah diakses, selanjutnya saya membuat beberapa **logika IF** untuk memeriksa pesan error, dan memberikan 'PESAN KESALAHAN' yang ingin ditampilkan kedalam variabel **\$pesan**.

Variabel **\$pesan** ini kemudian akan disisipkan di awal tag form.

Untuk mengujinya, silahkan anda mencoba **form.php** tersebut dan klik tombol 'PROSES DATA' tanpa memberikan input apa-apa di dalam form, dan sekarang pesan kesalahan akan tampil di halaman **form.php**.



Anda bisa mengeksplorasi penggunaan **fungsi header()** ini lebih jauh. Misalkan jika isian form telah sesuai, maka anda bisa meneruskan user ke halaman lain, seperti **sukses.php**, sehingga pengaturan kode PHP lebih rapi dimana halaman **proses.php** murni digunakan untuk penanganan proses form. Dari halaman **proses.php** ini kita bisa 'MEMBAWA' user ke halaman lain yang khusus menangani hal tersebut.

7 FILE DAN DIRECTORY

7.1 PENANGANAN FILE

Dalam management file dan direktori, PHP menyediakan lebih dari 70 fungsi. Beberapa fungsi utama yang berhubungan dengan management file (create, write, append, dan delete), antara lain :

Membuka dan Membuat File

```
fopen ($namafile, $mode);
```

Keterangan :

\$namafile merupakan nama file yang akan dibuat, sedangkan \$mode merupakan mode akses file. Mode akses file yang bisa digunakan yaitu :

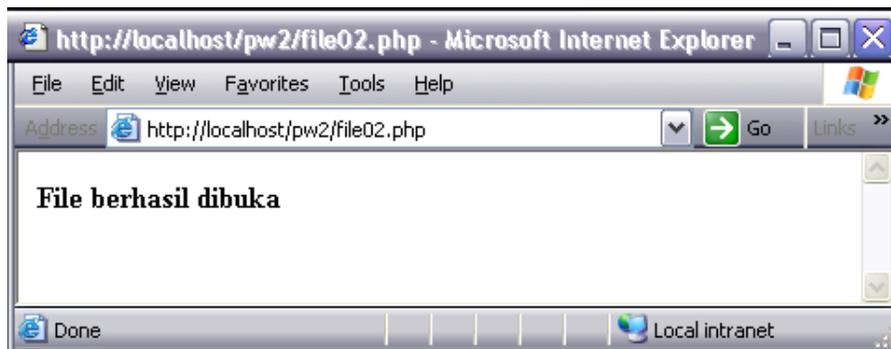
Mode	Keterangan
r	Hanya untuk baca file, pointer berada di awal file
r+	Untuk baca dan tulis file, pointer berada di awal file
w	Hanya untuk tulis file, isi file lama dihapus, jika file belum ada maka akan di-create
w+	Untuk baca dan tulis file, isi file lama dihapus, jika file belum ada maka akan di-create
a	Hanya untuk menambahkan isi file, pointer berada di akhir file, jika file belum ada maka di-create
a+	Untuk membaca dan menambahkan isi file, pointer berada di akhir file, jika file belum ada maka di-create

```
<?php
$namafile = "data.txt";
$handle = fopen ($namafile, "r"); if (!$handle) {
echo "<b>File tidak dapat dibuka atau belum ada</b>";
} else {
echo "<b>File berhasil dibuka</b>";
}
fclose($handle);
?>
```



Gambar 7.1 Tampilan Program 7.1

```
<?php
$namafile = "data.txt";
$handle = fopen ($namafile, "w");
if (!$handle) {
echo "<b>File tidak dapat dibuka atau belumada</b>";
} else {
echo "<b>File berhasil dibuka</b>";
}
fclose($handle);
?>
```



Gambar 7.2 Tampilan Program 7.2

Menulis ke File

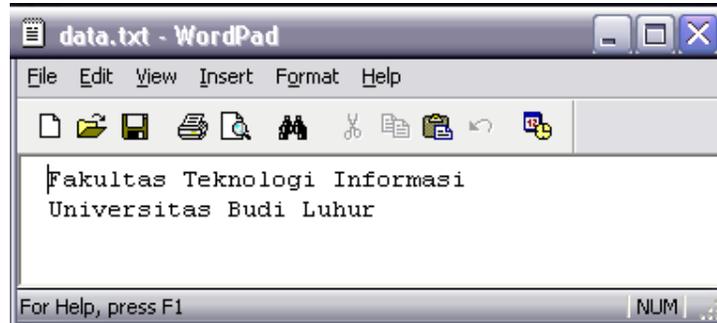
```
fwrite ($handle, $isi); fputs ($handle, $isi);
```

```

<?php
$namafilename = "data.txt";
$handle = fopen ($namafilename, "w");
if (!$handle) {
echo "<b>File tidak dapat dibuka atau belum ada</b>";
} else {
fwrite ($handle, "Fakultas Teknologi Informasi\n"); fputs ($handle,
"Universitas Budi Luhur\n");
//file_put_contents ($namafilename, "Jakarta"); echo "<b>File berhasil
ditulis</b>";
}
fclose($handle);
?>

```

Untuk menulis ke file, dapat menggunakan fungsi **fwrite()** dan **fputs()**. Variabel **\$handle** merupakan mode pengaksesan file, dimana mode yang digunakan harus bisa menulis ke file. Contohnya mode **w** dan **a**. Selain itu, kita juga bisa menggunakan fungsi **file_put_contents()** dengan parameter berupa nama file dan isi yang akan ditambahkan. Jika program di atas dijalankan, maka isi file **data.txt** akan berisi seperti pada gambar 7.3 (buka dengan WordPad).



Gambar 7.3. File **data.txt** dibuka dengan WordPad

Membaca Isi File

```

fread ($handle [, $panjang]); fgets ($handle [, $panjang]);

```

Contoh

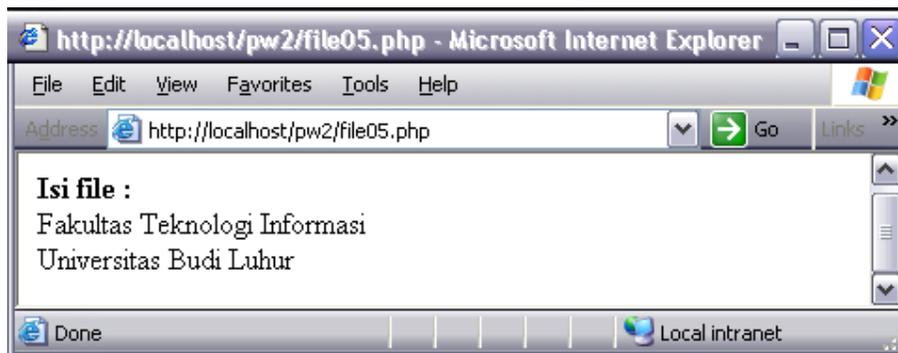
```
<?php
$namafile = "data.txt";
$handle = fopen ($namafile, "r"); if (!$handle) {
echo "<b>File tidak dapat dibuka atau belum ada</b>";
} else {
$isi = fgets ($handle, 2048);
//$isi2 = fread ($handle, 20); echo "Isi 1 : $isi<br>";
//echo "Isi 2 : $isi2<br>";
}
fclose($handle);
?>
```

Untuk membaca isi file, bisa menggunakan fungsi **fgets()** dan **fread()**. Fungsi **fgets()** akan membaca isi file tiap baris. Sedangkan fungsi **fread()** akan membaca isi file setiap jumlah byte tertentu. Pada program di atas, baris ke-8 akan membaca file setiap 20 byte atau 20 karakter.

Contoh lain

```
<?php
$namafile = "data.txt";
$handle = fopen ($namafile, "r"); if (!$handle) {
echo "<b>File tidak dapat dibuka atau belum ada</b>";
} else {
echo "<b>Isi file : </b><br>";
while ($isi = fgets ($handle, 2048)) { echo "$isi<br>";
}
}
fclose($handle);
?>
```

Program di atas akan menampilkan keseluruhan isi file, dimana isi file akan dibaca tiap baris. Pembacaan dilakukan dari baris pertama sampai baris terakhir.



Gambar 7.4. Tampilan Program 7.5

```
<?php
$namafile = "data.txt";
$handle = @fopen($namafile, "r"); if ($handle) {
while (!feof($handle)) {
$buffer = fgets($handle, 4096); echo $buffer."<br>";
}
fclose($handle);
}
?>
```

Program di atas akan menampilkan keseluruhan isi file, dimana isi file akan dibaca tiap baris. Pembacaan dilakukan selama pointer file belum sampai di end-of-file. Fungsi feof() pada baris ke-5 berfungsi untuk memeriksa pointer file sudah berada di akhir file atau belum.

Menutup File

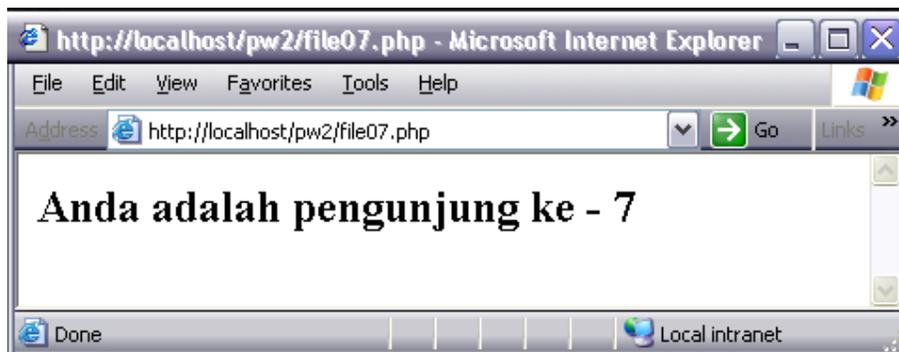
```
fclose ($handle);
```

Contoh lain

```
<?php
$counter_file="counter.txt";
if (!file_exists ($counter_file)) { fopen ($counter_file, "w");
}
$file = fopen($counter_file,"r");

$counter = fread($file,10); fclose($file);

$counter++;
echo "<h2>Anda adalah pengunjung ke - $counter</h2>";
$file = fopen($counter_file, "w"); fwrite($file,$counter);
fclose($file);
?>
```



Gambar 7.5. Tampilan Program 7.7

7.2 MEMBUAT DAN MENGHAPUS DIREKTORI

7.2.1 MEMBUAT DIREKTORI BARU

```
<?php
$dir = "images"; //nama direktori
$cek = mkdir ($dir); if ($cek) {
echo "Direktori <b>$dir</b> berhasil dibuat";
} else {
echo "Direktori <b>$dir</b> gagal dibuat";
}
?>
```

Program di atas akan membuat direktori bernama “images” di direktori tempat program 7.8 disimpan. Fungsi untuk membuat direktori baru dalam PHP adalah **mkdir()**. Jika direktori yang akan dibuat berada di luar tempat program disimpan, maka sertakan *path* direktori secara lengkap.

7.2.2 MENGHAPUS DIREKTORI

```
<?php
//buat direktori
$dir = "include";
$cek = mkdir ($dir); if ($cek) {
//nama direktori
echo "Direktori <b>$dir</b> berhasil dibuat";
} else {
echo "Direktori <b>$dir</b> gagal dibuat";
}
//hapus direktori
$del = rmdir ($dir); if ($del) {
echo "<br>Direktori <b>$dir</b> berhasil dihapus";
} else {
echo "<br>Direktori <b>$dir</b> gagal dihapus";
}
?>
```

Program di atas akan membuat direktori baru bernama “include” (lihat baris ke-4). Setelah itu, direktori akan dihapus dengan fungsi **rmdir()** pada baris ke-10. Fungsi **rmdir()** hanya bisa menghapus direktori kosong. Jika direktori ada isinya maka, gunakan fungsi rekursif untuk menghapus isi direktori. Lihat contoh di PHP Manual.

```
<?php
$dir = "images";
if ($handle = opendir($dir)) {
while (false !== ($file = readdir($handle))) { if ($file !=
"." && $file != "..") {
echo "$file<br>";
}
}
closedir($handle);
}
?>
```

Program 7.10 di atas akan menampilkan di layar, semua file dan direktori yang berada dalam direktori "images". Fungsi **opendir()** pada baris ke-3 adalah untuk membuka direktori. Sedangkan fungsi **readdir()** akan membaca file dalam direktori satu per satu secara urut. Fungsi **closedir()** pada baris ke-9 adalah untuk menutup pembukaan direktori.

7.3 MANIPULASI FILE

Beberapa Fungsi yang berhubungan dengan informasi dan manipulasi file, terdapat dalam tabel sebagai berikut :

Fungsi	Keterangan
file_exists(\$file)	Memeriksa apakah \$file ada atau tidak
is_file(\$file)	Memeriksa apakah \$file adalah file atau bukan.
is_dir(\$file)	Memeriksa apakah \$file adalah direktori atau bukan.
is_executable(\$file)	Memeriksa apakah \$file termasuk file yang bisa dijalankan secara langsung atau bukan.
is_writable(\$file)	Memeriksa apakah \$file termasuk file yang bisa ditulis atau diedit.
is_readable(\$file)	Memeriksa apakah \$file termasuk file yang bisa dibaca atau tidak.
fileatime(\$file)	Menghasilkan waktu akses terakhir file (unix timestamp).
filectime(\$file)	Menghasilkan waktu pembuatan file (unix timestamp).
filemtime(\$file)	Menghasilkan waktu modifikasi terakhir file (dalam unix timestamp).
filesize(\$file)	Menghasilkan ukuran besar file (dalam byte).
filetype(\$file)	Menghasilkan jenis file.

Contoh

```
<?php
$filename = "data.txt";

if (file_exists($filename)) { echo "File $filename ADA";
} else {
echo "File $filename TIDAK ADA";
}
?>
```

```
<?php
$file = "data.txt";
```

```

if (is_file($file)) {
echo "File <b>$file</b> adalah FILE<br>";
//
if (is_executable ($file)) {
echo "File <b>$file</b> bisa dijalankan langsung (executable)<br>";
    secara
} else {
echo "File <b>$file</b> TIDAK executable<br>";
}
//
if (is_writable ($file)) {
echo "File <b>$file</b> bisa ditulis / diedit<br>";
} else {
echo "File <b>$file</b> diedit<br>";
}    TIDAK bisa ditulis /
//
if (is_readable ($file)) {
echo "File <b>$file</b> bisa dibaca<br>";
} else {
echo "File <b>$file</b> TIDAK bisa dibaca<br>";
}
//
echo "Akses terakhir file <b>$file</b> = ". date("d-m-Y H:i:s.",
fileatime($file)). "<br>";
echo "File <b>$file</b> dibuat = ". date("d-m-Y H:i:s.",
filectime($file)). "<br>";
echo "Modifikasi terakhir file <b>$file</b> = ". date("d- m-Y
H:i:s.", filemtime($file)). "<br>";
echo "Ukuran file <b>$file</b> = ". filesize($file). " bytes<br>";
echo "Jenis file <b>$file</b> = ". filetype($file). "<br>";
} else if (is_dir($file)){
echo "File <b>$file</b> adalah DIREKTORI<br>";
} else {
echo "File <b>$file</b> tidak dikenal<br>";
}
?>

```

Contoh

```

<?php
$file = "data.txt";
$filebaru = "databaru.txt";
$filebaru2 = "databaru2.txt"; if (copy ($file, $filebaru)) {
echo "File <b>$file</b> berhasil dicopy menjadi
<b>$filebaru</b>. <br>";
}
if (rename ($filebaru, $filebaru2)) {
echo "File <b>$filebaru</b> berhasil direname menjadi
<b>$filebaru2</b>. <br>";
}
if (unlink ($filebaru2)) {
echo "File <b>$filebaru2</b> berhasil dihapus. <br>";
}

```

?>

Program 7.13 di atas terdapat beberapa fungsi untuk meng-copy, rename dan delete file. Fungsi **copy()** pada baris ke-5 akan menduplikasi file \$file menjadi file \$filebaru. Selanjutnya fungsi **rename()** akan mengganti nama file

\$filebaru menjadi \$filebaru2. Fungsi **unlink()** pada baris ke-11 akan menghapus file \$filebaru2.

7.4 UPLOAD FILE

PHP, sebagai server-side-scripting, sangat memungkinkan untuk menangani upload file ke server. Ada beberapa hal yang perlu diperhatikan dalam upload file ini, yaitu :

Pada Form HTML harus ditambahkan atribut :

```
ENCTYPE="multipart/form-data"
```

Form inputan upload file dapat menggunakan tag <input> dengan value atribut TYPE="FILE".

Untuk menangani inputan, PHP menyediakan sebuah variabel array global yaitu \$_FILES. Index dari variabel ini antara lain:

\$_FILES['file']['name'] : Nama asli dari file yang diupload

\$_FILES['file']['tmp_name'] : Nama temporary file yang diupload

\$_FILES['file']['size'] : Ukuran file asli (dalam byte)

\$_FILES['file']['type'] : MIME type file yang diupload

Destination folder file upload harus bisa writable (bisa diakses), biasanya dengan permission 777 atau 775.

```
<html>
<head><title>Upload File</title></head>
<body>
<FORM ACTION="upload.php" METHOD="POST"
ENCTYPE="multipart/form-data">
Upload File : <input type="file" name="file"><br>
<input type="submit" name="Upload" value="Upload">
</FORM>
</body>
</html>
```

Contoh Lain

```
<?php
if (isset($_POST['Upload'])) {
$dir_upload = "images/";
$nama_file = $_FILES['file']['name'];
//
if (is_uploaded_file($_FILES['file']['tmp_name'])) {
$cek = move_uploaded_file ($_FILES['file']['tmp_name'],
$dir_upload.$nama_file); if ($cek) {
die ("File berhasil diupload");
} else {
die ("File gagal diupload");
```

```
}  
}  
}  
?>
```

Program di atas merupakan program sederhana penanganan upload file. Fungsi **is_uploaded_file()** pada baris ke-6 akan mengupload file yang telah dipilih melalui form pada program 7.14 ke folder temporary (sementara). Selanjutnya pada baris ke-7, file yang sudah diupload ke folder temporary akan dipindahkan (move) ke direktori yang diinginkan menggunakan fungsi **move_uploaded_file()**. Lihat baris ke-7!



8 SESSION DAN COOKIES

Session merupakan hal yang cukup penting dalam aplikasi berbasis web. Dengan session memungkinkan programmer menyimpan informasi user secara semi-permanen, artinya selama masa tertentu informasi akan tersimpan. Penyimpanan isi variabel session berada di server, jadi tidak bisa diakses secara langsung oleh client.

Dalam aplikasi berbasis web, session banyak digunakan sebagai autentifikasi login. Dengan session memungkinkan programmer mengatur siapa saja yang bisa mengakses suatu halaman. Misalnya saja, untuk melihat halaman kotak surat pada email, kita harus login terlebih dahulu. Dalam proses login antara lain akan terjadi pembuatan suatu session yang akan dibawa oleh user di setiap halaman. Di halaman kotak surat, session tersebut akan diperiksa. Jika session benar maka user dipersilahkan membuka halaman kotak surat, namun jika salah maka user tidak bisa membuka halaman kotak surat dan biasanya akan diminta untuk login terlebih dahulu. Itulah sebabnya, user tidak bisa mengakses halaman kotak surat secara langsung tanpa melakukan login.

Dalam penanganan session terdapat beberapa proses yang perlu diperhatikan :

- Proses pembuatan session
- Proses pemeriksaan session
- Proses penghapusan session

Selanjutnya bagaimana session itu sendiri dijalankan? Agar proses penyimpanan dalam session berjalan, PHP melakukan beberapa hal berikut ini :

1. PHP meng-generate (membentuk) sebuah ID session.

2. ID session ini merupakan sejumlah deret angka random yang unik untuk setiap user dan hampir tidak mungkin bisa ditebak. ID session disimpan oleh PHP di dalam variabel sistem PHP dengan nama PHPSESSID
3. PHP menyimpan nilai yang akan Anda simpan dalam session di dalam file yang berada di server.
4. Nama file tempat penyimpanan session tersebut sesuai (sama) dengan ID session. File disimpan dalam suatu direktori yang ditunjukkan oleh session.save_path dalam file php.ini.
5. PHP melempar ID session ke setiap halaman.
6. PHP mengambil nilai session dari file session untuk setiap halaman session.

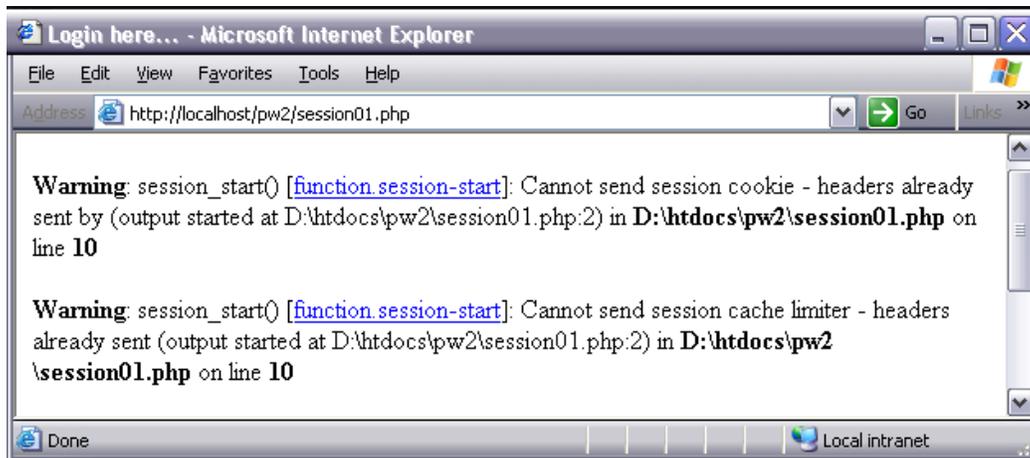
```

<?php
/*****
Nama file : session01.php
Halaman ini merupakan halaman contoh penciptaan session.
Perintah session_start() harus ditaruh di perintah pertama
tanpa spasi di depannya. Perintah session_start() harus ada
pada setiap halaman yang berhubungan dengan session
*****/
session_start();
if (isset ($_POST['Login'])) {
$user = $_POST['user'];

$pass = $_POST['pass'];
//periksa login
if ($user == "achmatim" && $pass = "123") {
//menciptakan session
$_SESSION['login'] = $user;
//menuju ke halaman pemeriksaan session
echo "<h1>Anda berhasil LOGIN</h1>";
echo "<h2>Klik <a href='session02.php'>di sini
(session02.php)</a>
untuk menuju ke halaman pemeriksaan session";
}
} else {
?>
<html>
<head>
<title>Login here...</title>
</head>
<body>
<form action="" method="post">
<h2>Login Here...</h2>
Username : <input type="text" name="user"><br>
Password : <input type="password" name="pass"><br>
<input type="submit" name="Login" value="Log In">
</form>
</body>
</html>
<? } ?>

```

Pada program di atas terdapat fungsi **session_start()** yang berada pada baris ke-9. Fungsi `session_start()` berfungsi untuk memulai sebuah session. Fungsi harus dipanggil pertama kali dalam suatu halaman PHP, artinya fungsi `session_start()` harus dipanggil sebelum sesuatu tampil di layar, walaupun hal itu hanya berupa spasi. Jika sebelum `session_start()` terdapat output yang ditampilkan maka akan terjadi error seperti pada gambar 9.1 berikut ini :



Gambar 9.1. Error yang terjadi karena sebelum `session_start()` sudah ada output
Selanjutnya, program 9.1 akan menampilkan sebuah form login sederhana di layar. Form terdiri dari form inputan username, password dan sebuah tombol login. Jika tombol login ditekan maka pemeriksaan kondisi pada baris 10 akan bernilai TRUE dan perintah yang ada di blok if akan dieksekusi. Baris 11 dan 12 merupakan perintah untuk mengambil nilai pada form inputan username dan password. Nilai username dan password tersebut diperiksa pada baris 14. Pemeriksaan username dan password ini, untuk aplikasi web yang sudah menggunakan database, bisa diganti dengan pemeriksaan username dan password ke tabel database.

Pada baris 16, terdapat perintah sebagai berikut :

```
$_SESSION['login'] = $user;
```

Baris perintah tersebut merupakan perintah untuk membuat session baru (create session) dimana nama session adalah "login" dan isi dari session adalah \$user.

\$_SESSION merupakan sebuah variabel array global yang didefinisikan oleh PHP, jadi variabel ini harus ditulis dengan huruf kapital (upper-case).

Selanjutnya, pada baris 18-20 akan ditampilkan pesan bahwa login berhasil dan juga sebuah link menuju ke halaman pemeriksaan session (session02.php).

```
<?php
/*****
Halaman ini merupakan contoh halaman pemeriksaan session. Pemeriksaan
session biasanya dilakukan jika suatu halaman memiliki
akses terbatas. Misalnya harus login terlebih dahulu.
*****/
session_start();
//pemeriksaan session
if (isset($_SESSION['login'])) { //jika sudah login
//menampilkan isi session
echo "<h1>Selamat Datang ". $_SESSION['login'] ."</h1>";
echo "<h2>Halaman ini hanya bisa diakses jika Anda sudah
login</h2>";
echo "<h2>Klik <a href='session03.php'>di sini
(session03.php)</a> untuk LOGOUT</h2>";
} else {
//session belum ada artinya belum login
die ("Anda belum login! Anda tidak berhak masuk ke halaman
ini.Silahkan login <a href='session01.php'>di sini</a>");
}
?>
```

Program 9.2 di atas diawali dengan pemanggilan fungsi **session_start()**. Hal ini berarti dalam halaman program ini terdapat operasi session. Pada baris 10 program di atas, terdapat pemeriksaan keberadaan variabel `$_SESSION['login']` dengan menggunakan fungsi **isset()**. Pemeriksaan kondisi ini akan bernilai TRUE jika `$_SESSION['login']` sudah terbentuk. Jika bernilai TRUE maka blok program baris 11-14 yang akan dieksekusi, dan sebaliknya, jika FALSE maka blok program baris 16-17 yang akan dijalankan. Blok program baris 11-14 hanya bisa diakses saat user sudah login dengan benar. Dengan kata lain, bagian tersebut merupakan bagian yang bersifat private (terbatas).

Untuk menampilkan isi variabel session dapat dilakukan seperti pada baris 12 program di atas. Cukup dengan meng-echo variabel `$_SESSION['login']`.

Jika user belum login (tidak melewati halaman login) atau langsung mengakses halaman ini, maka akan ditampilkan pesan bahwa user tidak berhak masuk ke halaman ini. Fungsi **die()** pada baris 17 digunakan untuk menampilkan suatu pesan sekaligus men-*terminate* program, artinya perintah selain die() akan diabaikan.

```

<?php
/*****
Halaman ini merupakan halaman logout, dimana kita menghapus
session yang ada.
*****/
session_start();
if (isset($_SESSION['login'])) {
unset ($_SESSION);
session_destroy();
//
echo "<h1>Anda sudah berhasil LOGOUT</h1>";
echo "<h2>Klik <a href='session01.php'>di sini</a> untuk
LOGIN kembali</h2>";
echo "<h2>Anda sekarang tidak bisa masuk ke halaman
<a href='session02.php'>session02.php</a> lagi</h2>";
}
?>

```

Program 9.3 merupakan contoh program penghapusan session. Penghapusan session biasanya digunakan ketika user melakukan logout. Proses penghapusan session dilakukan dengan memanggil fungsi `unset()` dan fungsi `session_destroy()`. Fungsi **`unset()`** digunakan untuk menghapus atau menghilangkan suatu variabel. Sedangkan **`session_destroy()`** digunakan untuk menghapus semua data yang berhubungan dengan session tersebut.

8.1 COOKIES

Seperti halnya session, cookies juga merupakan sebuah konsep penyimpanan informasi user. Hanya saja, jika session tempat penyimpanan berada di server, cookies berada di client. Oleh karena itu, konsep cookies sebaiknya jangan digunakan untuk menyimpan informasi login user seperti username, password dsb. Selain user bisa melihat informasi yang disimpan, user juga bisa men-disable cookies itu sendiri. Jika cookies di-disable, maka program yang memanfaatkan cookies tentunya tidak akan berjalan dengan baik.

Cookies sendiri biasanya dipakai dalam aplikasi shopping cart. Biasa digunakan untuk menyimpan sementara, produk-produk yang dipilih oleh pengunjung pada saat berbelanja.

Dalam penanganan cookies juga terdapat beberapa proses yang perlu diperhatikan :

- Proses pembuatan cookies
- Proses pemeriksaan cookies
- Proses penghapusan cookies

```

<?php
$value = 'achmatim';
$value2 = 'Achmad Solichin';
setcookie("username", $value);
setcookie("namalengkap", $value2, time()+3600); /* expire in 1
hour */
echo "<h1>Ini halaman pengesetan cookie</h1>";
echo "<h2>Klik <a href='cookie02.php'>di sini</a> untuk
pemeriksaan cookies</h2>";
?>

```

```

<?php
if(isset($_COOKIE['username'])) {
echo "<h1>Cookie 'username' ada. Isinya : " .
$_COOKIE['username'];
} else {
echo "<h1>Cookie 'username' TIDAK ada.</h1>";
}
if(isset($_COOKIE['namalengkap'])) {
echo "<h1>Cookie 'namalengkap' ada. Isinya : " .
$_COOKIE['namalengkap'];
} else {
echo "<h1>Cookie 'namalengkap' TIDAK ada.</h1>";
}
echo "<h2>Klik <a href='cookie01.php'>di sini</a> untuk
penciptaan cookies</h2>";
echo "<h2>Klik <a href='cookie03.php'>di sini</a> untuk
penghapusan cookies</h2>";
?>

```

Contoh Lainnya

```

<?php
// set the expiration date to one hour ago
setcookie ("username", "", time() - 3600);
setcookie ("namalengkap", "", time() - 3600);
echo "<h1>Cookie Berhasil dihapus.</h1>";
echo "<h2>Klik <a href='cookie01.php'>di sini</a> untuk
penciptaan cookies</h2>";
echo "<h2>Klik <a href='cookie02.php'>di sini</a> untuk
pemeriksaan cookies</h2>";
?>

```

9 OOP REVIEW

9.1 ENKAPSULASI

9.2 VARIABEL \$THIS

9.3 PROPERTY DAN METHOD DALAM CLASS

9.4 AUTOLOAD