MODUL PEMROGRAMAN WEB

REGULAR EXPRESSION

Rajif Agung Yunmar, S.Kom

STMIK “AMIKOM” Yogyakarta

2011

DAFTAR ISI

REGULAR EXPRESSION

Regular Expression (RegEx) merupakan salah satu implementasi dari operasi pencocokan pola (*Pattern Recognation*) untuk sebuah text atau string. Dengan regex kita dapat mencari text yang sesuai dengan sebuah pola atau aturan tertentu, melakukan validasi terhadap input data, dan lain lain.

Regular Expression terdiri dari string yang merupakan kombinasi antara karakter normal, karakter meta tertentu dan *meta sequences*. Dalam implementasi dari kombinasi ini, karakter normal mewakili karakter itu sendiri. Sedangkan Meta karakter dan *meta sequences* adalah karakter atau sequences yang merepresentasikan maksud tertentu. Misalnya: kuantitas, lokasi, atau tipe dari karakter tertentu.

1. Teori Dasar

Teori dasar mengenai regex dapat dibawa ke lingkungan kerja manapun. Regex dapat diterapkan pada banyak bahasa pemrograman. Mulai dari Perl, PHP, Python, Java, JavaScript hingga VB. Tidak hanya itu saja ia juga dapat diterapkan pada Unix melalui utility grep dan awk, begitu juga dengan Windows dan OS lain dengan berbagai program lainnya.

Misalnya: regex dapat digunakan pada editor teks (seperti vi dan joe), file konfigurasi berbagai daemon dan utilitas (seperti procmail, exim, dan Apache), bahkan di aplikasi seperti Microsoft Word dan Borland Delphi pun tak luput dari penggunaan regex.

1.1 Wildcard

Wildcard merupakan sebuah bentuk primitif dari regex dan banyak digunakan pada DOS dan Linux shell. Sedangkan pada lingkungan Unix, wildcard lebih dikenal dengan istilah globbing. Wildcard adalah sebuah string pola yang digunakan untuk mencocokkan sekumpulan file atau direktori yang sesuai. Perhatikan contoh berikut ini:

String Pola	Keterangan
`*.doc`	Cocok dengan semua file dengan ekstensi *.doc. Karakter * disini berarti cocok dengan satu atau deretan karakter apa saja.
`????.txt`	Cocok dengan semua file dengan ekstensi .txt dan memiliki nama file yang terdiri dari empat karakter.

1.2 Karakter Meta

Beberapa karakter dalam sebuah pola regex akan diartikan secara khusus dan bukan mewakili karakter sebenarnya. Karakter ini disebut sebagai karakter meta. Seperti pada contoh sebelumnya dimana karakter * dan ? pada wildcard tidak diartikan sebagai karakter literal (karakter dengan arti sebenarnya) bintang dan tanda tanya, melainkan karakter dengan arti yang khusus.

1.2.1 Pemilihan

Karakter meta pemilihan sering kali disebut juga dengan karakter meta alternasi. Karakter meta pemilihan diwakili oleh karakter `|` (garis lurus vertikal) dan dibaca sebagai “atau”. Gunanya untuk memilih satu dari dua atau lebih alternatif yang disediakan. Contoh :

- `aku|kamu` akan cocok dengan **aku** atau **kamu**, tetapi tidak dengan **dia**.
- `sate|bakso|siomay` akan cocok dengan **sate**, **bakso** atau **siomay**. Tetapi tidak dengan **batagor**.

1.2.2 Pengelompokan

Pengelompokan diwakili oleh karakter `(` dan `)`, yang digunakan untuk mengelompokkan set aturan. Pada umumnya karakter meta pengelompokan digunakan bersamaan dengan karakter meta lain. Contoh:

- `satria (baja hitam|pembela kebetulan)` akan cocok dengan **satria baja hitam** atau **satria pembela kebetulan**.
- `garuda (muda|di dadaku)` akan cocok dengan **garuda muda** atau **garuda di dadaku**.

1.2.3 Set Karakter

Set karakter diwakili oleh karakter `[` dan `]`, pada dasarnya juga digunakan untuk pemilihan layaknya karakter meta `|`. Namun, set karakter ini mempunyai fasilitas syntax rentang dan negasi. Contoh dari syntax rentang adalah `[m-n]`, yang akan cocok dengan karakter mulai dari `m` hingga `n`. Contoh dari syntax negasi adalah `[^m]`, yang akan cocok dengan semua karakter kecuali karakter huruf `m`. Perhatikan contoh dari aturan set karakter berikut ini :

- `bat[aiuo]k` akan cocok dengan **batak**, **batik**, **batuk** atau **batok**.
- `bat(a|i|u|o)k` sama dengan pola atau sebelumnya, namun menggunakan kombinasi karakter meta pemilihan dan pengelompokan.
- `[0-9]` akan cocok dengan angka 0 sampai 9.

- `[A-EG-Z]` akan cocok dengan semua huruf besar kecuali F.
- `[0-9][0-9]` akan cocok dengan 00 sampai 99 (100 kombinasi).
- `[012][0-9]` akan cocok dengan 00 sampai 29 (30 kombinasi).
- `[012][0-9]|30` akan cocok dengan '00 sampai 29' atau 'angka 30' (31 kombinasi).
- `sem([ui]|bilan)` akan cocok dengan **semu**, **semi** atau **sembilan**. Tetapi tidak dengan **semubilan**, **semibilan** atau **semuibilan**.

1.2.4 Optional

Karakter meta optional diwakili oleh simbol `?` (tanda tanya). Karakter meta optional dalam regex mempunyai arti yang berbeda dengan simbol `?` pada wildcard. Karakter meta optional dalam regex artinya huruf atau kelompok aturan yang berada pada sebelah kiri tanda tanya (`?`) bersifat optional. Dapat juga dibaca “boleh ada atau boleh juga tidak”. Contoh :

- `silah?kah` cocok dengan **silakan** atau **silahkan**.
- `(silah)?kan` akan cocok dengan **silahkan** atau **kan** saja.
- `advi([sc]es?|sory)` akan cocok dengan **advise**, **advises**, **advice**, **advices** atau **advisory**. Jika dibaca, pola string regex ini berbunyi : deretan huruf `advi` diikuti dengan salah satu dari pilihan :
  a. Huruf `s` atau `c`, diikuti huruf `e`. Kemudian boleh diikuti dengan huruf `s`.
  b. Deretan huruf `sory`.

1.2.5 Titik

Titik atau dot adalah simbol dalam regex yang cocok dengan semua karakter tunggal. Contoh :

- `bat.k` akan cocok dengan **batik**, **batok**, **bat+k**, **bat8k**, dsb. Namun tidak cocok dengan **batruk** (karena **ru** adalah 2 karakter) atau **batk** (nol karakter).
- `bat.?k` sama dengan pola sebelumnya. Namun, pola ini cocok dengan **batk** (nol karakter) karena karena terdapat karakter meta optional (`?`) setelah tanda titik.
- `b...k` akan cocok dengan banyak kata dan kombinasi. Terdiri dari 5 karakter yang diawali dengan huruf `b` dan diakhiri dengan huruf `k`. Misalnya: **batuk**, **bebek**, **bilik**, **batak**, dsb.

1.2.6 Pengulangan

Karakter meta perulangan diwakili oleh simbol `*` atau `+`. Jika pada karakter meta

pemilihan (`?`) dapat diartikan “boleh ada boleh tidak” atau “nol atau satu”, maka karakter meta `*` dapat diartikan “nol atau lebih” dan karakter meta `+` dapat diartikan “satu atau lebih” dari karakter atau set aturan yang tepat berada pada sebelah kiri karakter meta pengulangan tersebut. Contoh :

- `[0-9]+` akan cocok dengan deretan angka berapapun.
- `.+` cocok dengan satu atau lebih karakter apapun. Namun tidak cocok dengan string kosong.
- `.*` cocok dengan karakter apapun, termasuk string kosong.
- `h?(ah|eh)+!*` akan cocok dengan **ah**, **heh**, **hah!**, **hehehe** atau **ahahahaha!!!**. Pola ini dapat dibaca sebagai berikut:
  a. Boleh diawali dengan huruf `h`.
  b. Diikuti dengan deretan huruf `ah` atau `eh`.
  c. Dan boleh diakhiri dengan lebih dari satu tanda seru (`!`).

1.2.7 Jangkar

Karakter meta jangkar diwakili oleh simbol `^` dan `$`. Masing-masing simbol tersebut dapat diartikan “harus diawal” dan “harus diakhir”. Karakter meta ini tidak melambangkan arti apapun, melainkan mensyaratkan *posisi* atau *penambatan* pola ke string yang ingin dicocokkan. Itulah sebabnya pasangan karakter meta ini disebut anchor atau jangkar. Contoh :

- `456` akan cocok dengan **456**, **1234567** atau **456789**. Karena ketiganya mengandung pola huruf **456**.
- `^456` akan cocok dengan **456**, **456789**. Tetapi tidak cocok dengan **1234567**, karena pola regex tersebut mensyaratkan huruf **456** ada didepan string yang dicocokkan.
- `^456$` akan cocok dengan **456**. Tetapi tidak dengan **1234567** dan **456789**.

1.2.8 Quantifier

Karakter meta quantifier menyatakan berapa rentang atau jumlah karakter yang diperbolehkan dari sebuah pola (satu atau kelompok karakter yang berada di sebelah kiri quantifier). Berikut ini adalah beberapa format dari quantifier:

- `X{m}` artinya set aturan `X` harus ada sebanyak `m` kali.
- `X{m,}` artinya set aturan `X` harus ada minimal sebanyak `m` kali.
- `X{,n}` artinya set aturan `X` boleh ada hingga terulang maksimal `n` buah.
- `X{m,n}` artinya set aturan `X` boleh ada dari minimal `m` buah hingga terulang

sebanyak maksimal `n` buah.

Berikut ini adalah beberapa contoh dari penerapan karakter meta quantifier :

- `[0-9]{4}` akan cocok deretan empat digit angka dimulai dari 0000 sampai dengan 9999.
- `[0-9]{1,4}` akan cocok dengan 0 sampai dengan 9999 (mulai dari bilangan 1 digit sampai 4 digit).
- `[0-9]{1,}` akan cocok dengan deretan digit angka.

Karakter meta pengulangan `?`, `+` dan `*` sebenarnya adalah jalan pintas dari syntax quantifier yang lebih umum. Dimana :

- Karakter meta `?` akan sama dengan quantifier `{0,1}`
- Karakter meta `+` akan sama dengan quantifier `{1,}`
- Karakter meta `*` akan sama dengan quantifier `{0,}`

1.2.9 Modifier

Dalam proses pencocokan pola, perilaku dari mesin regex dapat diubah dengan modifier. Terdapat beberapa modifier yang dikenal dalam mesin regex Perl-compatible. Diantaranya adalah sebagai berikut :

a. Modifier `i` (IGNORE_CASE)

   Jika kita menggunakan modifier ini, maka mesin regex tidak akan membedakan antara huruf besar dan kecil. Artinya pola [a-z] dengan modifier `i` akan dianggap sama dengan pola [a-zA-Z]. Modifier ini bermanfaat untuk mempersingkat pola, jika kita menginginkan pencocokan yang tidak membedakan huruf besar dan kecil.

b. Modifier `s` (SINGLE_LINE)

   Masih ingat dengan karakter meta `.` (titik)? Pada dasarnya, karakter meta titik akan cocok dengan string atau karakter apapun. Kecuali karakter *new line* (\n) atau *enter*. Contoh :

   `Selamat.+` akan cocok dengan **Selamat Datang** namun tidak dengan **Selamat\nDatang** (deretan karakter **Selamat** diikuti deretan karakter **Datang** pada baris selanjutnya). Ini dikarenakan mesin regex menganggap akhir baris sebagai akhir dari sebuah string yang hendak dicocokkan. Sehingga sub pola `.+` tidak akan cocok dengan string **Selamat\nDatang**, dimana setelah string **Selamat** tidak ada lagi karakter pada baris pertama string.

   Dengan menggunakan modifier `s`, karakter meta `.` (titik) akan cocok dengan *new*

line, sehingga mesin regex akan menelan semua sisa string hingga **\nDatang**. Dengan kata lain, dengan modifier `s`, setiap string akan dianggap terdiri dari satu baris saja.

c. Modifier `m` (MULTIPLE_LINE)

Muskipun dinamakan *multiple line*, modifier ini bukanlah kebalikan dari modifier `s`. Bahkan kedua modifier ini dapat dipakai secara bersamaan tanpa saling bentrok satu sama lain. Contoh :

Tanpa ada modifier `m`, `^Datang` tidak akan cocok dengan string **Selamat\nDatang**. Karena karakter meta jangkar `^` mensyaratkan deretan karakter **Datang** ada pada awal string, dalam hal ini awal dari string adalah **Selamat**.

Dengan modifier `m`, pola tersebut akan cocok karena tiap awal dan akhir baris dianggap sebagai ujung yang bisa cocok dengan karakter meta jangkar. Contoh lain, `Selamat$` akan cocok di mode `m` tapi tidak tanpa `m`. Dengan kata lain, dalam mode `m` setiap baris dalam string akan dianggap memiliki awal dan ujung string masing-masing.

d. Modifier `g` (GLOBAL_MATCH)

Terutama hanya bekerja pada bahasa pemrograman Perl. Dengan modifier ini, pencocokan berikutnya akan dilanjutkan dari posisi terakhir yang ditemukan, tidak diulang dari posisi awal string. Terutama berguna jika ingin melakukan iterasi string dari awal hingga akhir.

1.2.10 Karakter Escape

Karakter escape diwakili oleh `\` (backslash, garis miring terbalik). Karakter meta escape digunakan untuk dua keperluan :

a. Menjadikan karakter meta yang tepat berada di belakang escape menjadi sebuah karakter literal. Contoh :

Kita ingin melakukan pencocokan bilangan desimal dengan pemisah berupa titik. Permasalahannya adalah titik digunakan sebagai karakter meta yang cocok dengan karakter apapun. Oleh karena itu digunakan karakter escape didepan karakter titik untuk menyatakan bahwa karakter titik dibelakang karakter escape adalah karakter literal titik yang sesungguhnya.

`[0-9]+(\.[0-9]*)?` akan cocok dengan **5**, **12.3**, **100.873**, dsb. Pola regex tersebut dapat dibaca sebagai berikut : “Deretan satu atau lebih digit angka yang

boleh diikuti dengan `.` (titik) dan deretan angka.

Bandingkan dengan pola regex `[0-9]+(.[0-9]*)?` (tanpa karakter escape didepan titik) yang akan cocok dengan **12p3**, **100x873**, dsb. Yang menjadikannya bukan bilangan desimal yang valid.

b. Digunakan untuk menyatakan karakter yang tidak bisa dinyatakan secara langsung. Contoh :
   - Pola `\t` untuk tab, `\n` untuk new line atau enter.
   - Pola `\xMN` dan `\OABC` yang masing-masing menyatakan karakter ASCII dengan kode hexadesimal (bilangan basis 16) dan octal (bilangan basis 8). Misalnya : `\x09` dalam hexadesimal untuk mewakili tab, `\0012` dalam octal untuk mewakili new line.

1.3 Character Classes

Character Classes adalah cara yang digunakan untuk mendefiniskan atau menentukan set karakter. *Character Classes* sering digunakan untuk mempersingkat atau menggantikan pola regex untuk kelompok string dengan maksud tertentu. Misalnya : pola `[0-9]` dapat digantikan oleh kelompok karakter `\d` saja.

1.3.1 Shorthands Class

Pola	Deskripsi
`\c`	Control Charakter.
`\s`	White space, `[\n\r\f\t ]`.
`\S`	Not white space, `[^\n\r\f\t ]`.
`\d`	Digit, `[0-9]`.
`\D`	Not digit, `[^0-9]`.
`\w`	Word, `[a-zA-Z0-9_]`.
`\W`	Not Word, `[^a-zA-Z0-9_]`.
`\xMN`	Hexadecimal Character.
`\OABC`	Octal Character.

1.3.2 POSIX Character Classes

Pola	Deskripsi
`[:upper:]`	Uppercase letters.
`[:lower:]`	Lowercase letters.

`[:alpha:]`	All letters (lower and upper case letters).
`[:alnum:]`	Digits and letters.
`[:digit:]`	Digits.
`[:xdigit:]`	Hexadecimal digits.
`[:punct:]`	Punctuation.
`[:blank:]`	Space and tab.
`[:space:]`	Blank characters.
`[:cntrl:]`	Control characters.
`[:graph:]`	Printed characters.
`[:print:]`	Printed characters and space.
`[:word:]`	Word (digits, letters and underscore).

1.3.3 Unicode Character Classes

Pola	Deskripsi
`\p{L}`	Letter.
`\p{Ll}`	Lowercase letters.
`\p{Lm}`	Modifier letters.
`\p{Lo}`	Letters, other. These have no case and are not considered modifiers.
`\p{Lt}`	Titlecase letters.
`\p{Lu}`	Uppercase letters.
`\p{C}`	Control codes and characters not in other categories.
`\p{Cc}`	ASCII and Latin-1 control characters.
`\p{Cf}`	Non-visible formatting characters.
`\p{Cn}`	Unassigned code points.
`\p{Co}`	Private use, such as company logos.
`\p{Cs}`	Surrogates.
`\p{M}`	Marks meant to combine with base characters, such as accent marks.
`\p{Mc}`	Modification characters that take up their own space. Examples include "vowel signs."
`\p{Me}`	Marks that enclose other characters, such as circles, squares, and diamonds.
`\p{Mn}`	Characters that modify other characters, such as accents and umlauts.
`\p{N}`	Numeric characters.
`\p{Nd}`	Decimal digits in various scripts.

`\p{Nl}`	Letters that are numbers, such as Roman numerals.
`\p{No}`	Superscripts, symbols, or non-digit characters representing numbers.
`\p{P}`	Punctuation.
`\p{Pc}`	Connecting punctuation, such as an underscore.
`\p{Pd}`	Dashes and hyphens.
`\p{Pe}`	Closing punctuation complementing \p{Ps}.
`\p{Pi}`	Initial punctuation, such as opening quotes.
`\p{Pf}`	Final punctuation, such as closing quotes.
`\p{Po}`	Other punctuation marks.
`\p{Ps}`	Opening punctuation, such as opening parentheses.
`\p{S}`	Symbols.
`\p{Sc}`	Currency.
`\p{Sk}`	Combining characters represented as individual characters.
`\p{Sm}`	Math symbols.
`\p{So}`	Other symbols.
`\p{Z}`	Separating characters with no visual representation.
`\p{Zl}`	Line separators.
`\p{Zp}`	Paragraph separators.
`\p{Zs}`	Space characters.

2. Implementasi

2.1 PHP

Didalam bahasa pemrograman PHP, terdapat dua fungsi yang sering digunakan untuk menjalankan regular expression. Pertama, fungsi dengan prefiks `ereg_*` dan `eregi_*` yang bergaya POSIX. Kedua, fungsi dengan prefix `preg_*` yang bergaya Perl (menggunakan pustaka PCRE, Perl Compatible Regular Expression).

Fungsi regex menggunakan PCRE mempunyai kelebihan tersendiri seperti optimasi regular expression, asersi, lihat ke belakang, lihat kedepan, dll. Kecuali modifier dan dan shortcut, teori dasar yang telah dijelaskan sebelumnya dapat berjalan pada kedua gaya regex. Melihat kelebihan dari fungsi regex dengan PCRE, maka pembahasan implementasi regex pada PHP akan difokuskan pada fungsi dengan prefix `preg_*`.

Berikut ini adalah beberapa fungsi PCRE yang terdapat di PHP :

- `array preg_match(str pola, str subjek[, array matches]);`

- `int preg_match_all(str pola, str subjek, array matches[, int order]);`
- `mixed preg_replace(mixed pola, mixed pengganti, mixed subjek[, int limit]);`
- `array preg_split(str pola, str subjek[, int limit[, int flags]]);`
- `string preg_quote(str s[, str kutip]);`
- `array preg_grep(str pola, array input);`

2.1.1 Penulisan

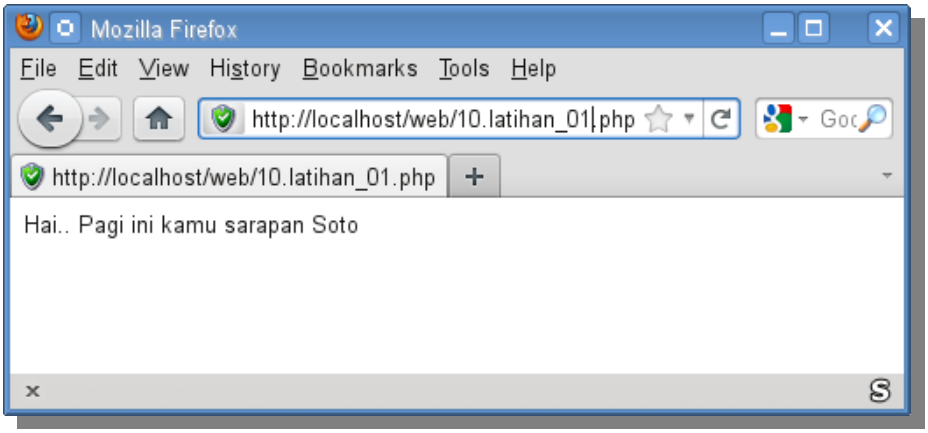
Pola regex ditulis sebagai string dan diletakkan diantara tanda pengapit `//` atau tanda pengapit yang dapat kita definisikan sendiri. Sedangkan modifier dituliskan setelah tanda pengapit terakhir (`/`).

2.1.2 Fungsi preg_match

Fungsi `preg_match` pada PHP sama dengan fungsi `m//` yang terdapat pada Perl. Namun tanpa dikenakan modifier `g` pada akhir pola. Contoh :

10.latihan_01.php

```php
<?php
$text = "Pagi ini saya sarapan Soto.";
preg_match("/(soto|siomay|sate)/i", $text, $matches);
echo "Hai.. Pagi ini kamu sarapan ".$matches[0];
?>
```



Program tersebut diatas dapat dijelaskan sebagai berikut :

- Program melalui fungsi `preg_match` akan mencocokkan string pola `(soto|siomay|sate)` dan modifier `i` (case insensitive) dengan karakter yang terdapat

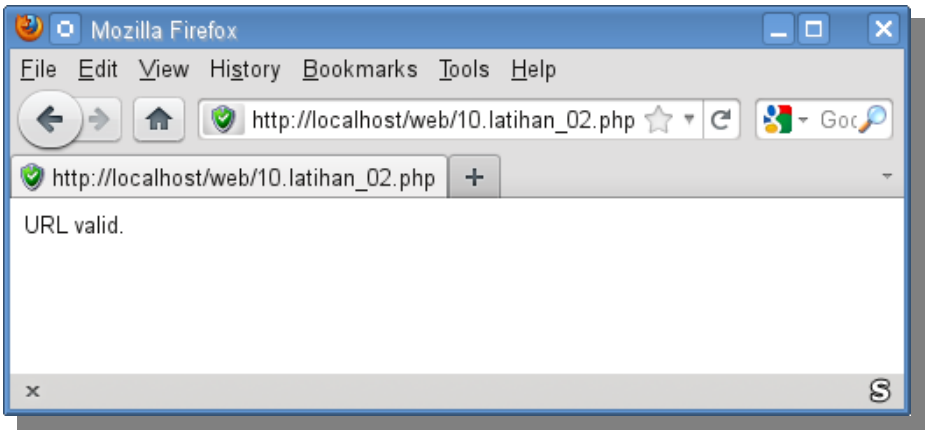
pada variabel `$text`.

- Jika terdapat item yang cocok dengan pola regex, item-item tersebut akan disimpan pada variabel `$matches` dalam bentuk array.
- Baris program selanjutnya akan menampilkan item yang cocok dengan pola regex ke browser pengguna.

10.latihan_02.php

```
<?php
$text = "http://amikom.ac.id/";

if(preg_match("/^http:\/\//i", $text)){
   echo "URL valid.";
} else {
   echo "URL tidak valid.";
}
?>
```



Contoh latihan program tersebut diatas digunakan untuk melakukan pengecekan terhadap format URL yang terdapat pada variabel `$text`. Jika URL yang terdapat pada variabel `$text` bernilai valid, maka fungsi `preg_match` akan mengembalikan nilai **`true`**, dan **`false`** jika sebaliknya.

Sekarang, cobalah mengganti format fungsi `preg_match("/^http:\/\//i", $text)` diatas menjadi `preg_match("@^http://@i", $text)`, kemudian amati dan analisa hasilnya. Adakah perbedaan?

2.1.3 Fungsi preg_match_all

PHP tidak mendukung modifier `g` seperti halnya fungsi regex pada Perl. Namun PHP menyediakan fungsi `preg_match_all` sebagai pengganti dari modifier ini. Fungsi ini biasa digunakan jika kita menginginkan pengambilan seluruh kelompok item yang cocok dengan

pola dalam satu text secara sekaligus. Perhatikan contoh program berikut :

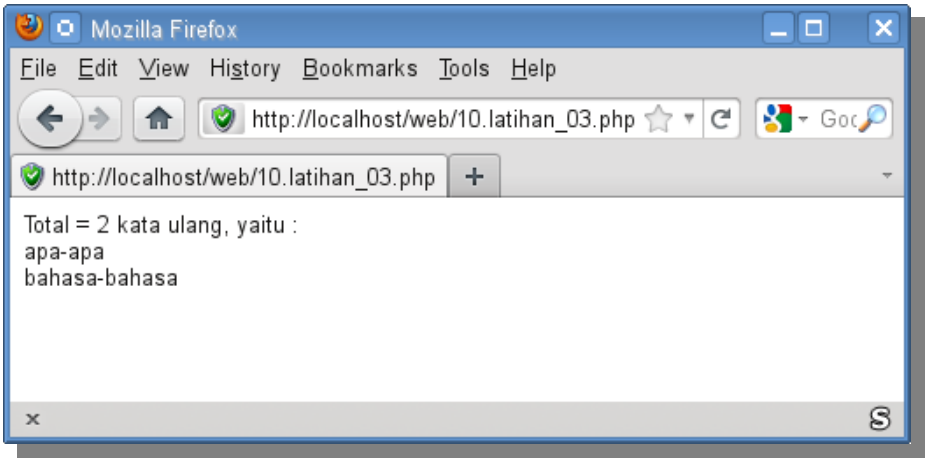
10.latihan_03.php

```php
<?php

$teks = "Dari awal saya bilang juga ndak apa-apa ".
        "mo Perl kek, PHP kek, Java kek ".
        "bahasa-bahasa laen kek, semua oke!";
$n = 0;
preg_match_all("/((\w+)-.+? )/", $teks, $matches, PREG_SET_ORDER);

echo "Total = ".count($matches)." kata ulang, yaitu :<br />";
for ($i=0; $i < count($matches); $i++) {
   echo $matches[$i][1]. "<br />";
   $n++;
}

?>
```



Bandingkan program diatas dengan cara mengganti fungsi `preg_match_all` dengan fungsi `preg_match`, kemudian amati dan lihat hasilnya.

2.1.4 Fungsi preg_replace

Fungsi `preg_replace` digunakan untuk mengganti string yang cocok dari pola regex dengan nilai yang ditentukan. Contoh :

10.latihan_04.php

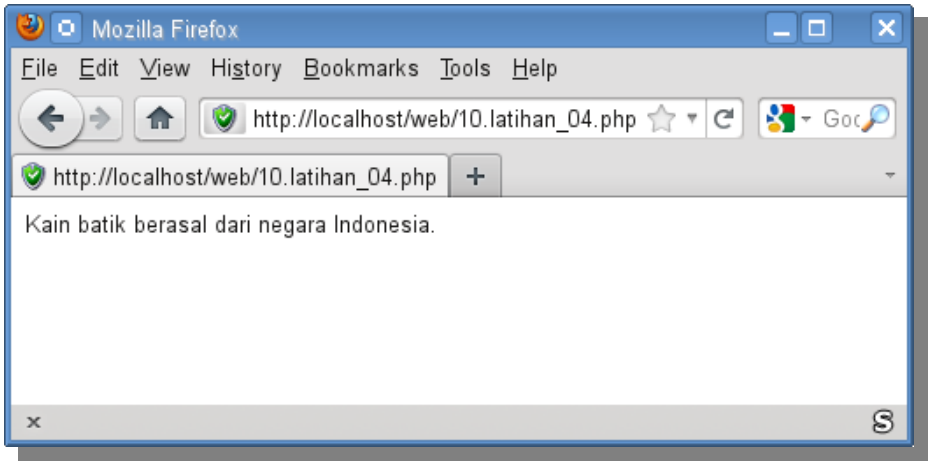
```php
<?php

$text = "Kain batik berasal dari negara Malaysia.";

preg_replace("/Malaysia/","Indonesia",$text);

echo $text;

?>
```



Program tersebut diatas menjalankan fungsi regex yang digunakan untuk mengganti kata Malaysia pada variabel `$text` menjadi Indonesia, kemudian nilainya disimpan kembali ke variabel `$text`.

2.2 JavaScript

Fungsi-fungsi regex telah didukung sejak JavaScript 1.2 yang terdapat pada browser Nescape 4.x dan Internet Explorer 4. Kecuali modifier `s` dan `m`, semua teori dasar yang disampaikan sebelumnya telah didukung penuh oleh JavaScript 1.3 keatas.

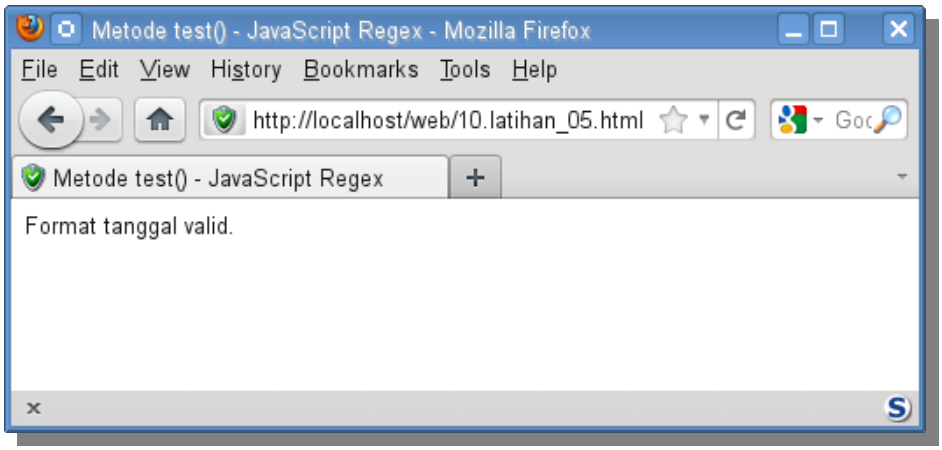
2.2.1 Metode Test

Metode `test()` hanya digunakan untuk mengetahui apakah format text yang akan dicocokkan bernilai valid. Jika cocok, metode ini akan mengembalikan nilai **`true`** dan **`false`** jika sebaliknya. Cara penulisan regex dengan metode `test()` ini mirip dengan metode PCRE, yaitu dengan cara menyisipkan pola regex diantara tanda `//` dan meletakkan modifiernya di belakang tanda pengapit terakhir (`/`). Contoh :

10.latihan_05.php

```
<html>
   <title>Metode test() - JavaScript Regex</title>
   <body>
      <script language="JavaScript">
         var date    = "31-12-2011";
         var pattern = /(\d{2})-(\d{2})-(\d{4})/;

         if(pattern.test(date) == true){
            document.write("Format tanggal valid.");
         } else {
            document.write("Format tanggal tidak valid.");
         }
      </script>
   </body>
</html>
```



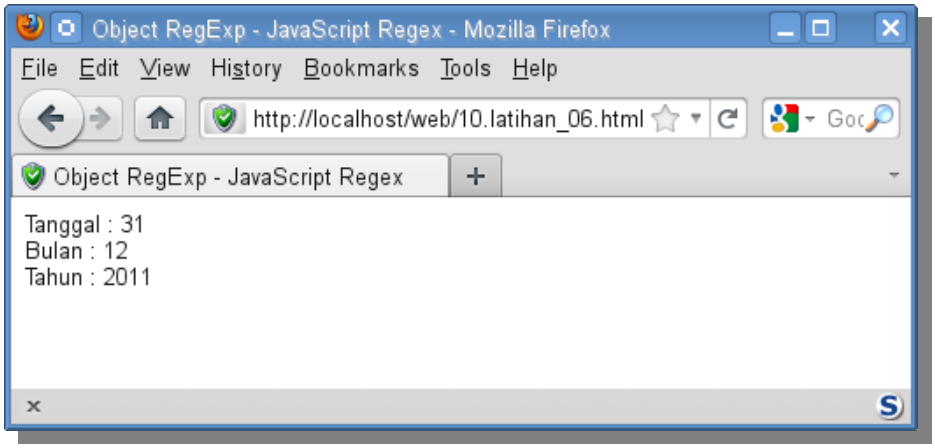
2.2.2 Object RegExp

Berbeda dengan metode `test()`, selain mencocokkan, object `RegExp` juga dapat digunakan untuk mengambil item-item yang cocok dengan pola yang ditentukan. Contoh :

10.latihan_06.php

```
<html>
   <title>Object RegExp - JavaScript Regex</title>
   <body>
      <script language="JavaScript">
         var date    = "31-12-2011";
         var pattern = new RegExp("(\\d{2})-(\\d{2})-(\\d{4})");

         if(pattern.exec(date) != null){
            document.write("Tanggal : " + RegExp.$1);
            document.write("<br />");
            document.write("Tanggal : " + RegExp.$2);
            document.write("<br />");
            document.write("Tanggal : " + RegExp.$3);
         }
      </script>
   </body>
</html>
```



Dengan hasil yang sama, program diatas juga dapat dituliskan dengan cara berikut :

10.latihan_07.php

```
<html>
   <title>Object RegExp - JavaScript Regex</title>
   <body>
      <script language="JavaScript">
         var date    = "31-12-2011";
         var pattern = new RegExp("(\\d{2})-(\\d{2})-(\\d{4})");

         var result = pattern.exec(date);

         if(result != null){
            document.write("Tanggal : " + result[1]);
            document.write("<br />");
            document.write("Tanggal : " + result[2]);
            document.write("<br />");
            document.write("Tanggal : " + result[3]);
         }
      </script>
   </body>
</html>
```

Kedua program diatas melakukan hal yang sama, yaitu mencocokkan pola menggunakan object RegExp, kemudian menampilkan item-item yang cocok dengan pola regex tersebut. Berdasarkan kedua program diatas pula, kita dapat mengambil item-item yang cocok dengan pola regex dengan dua cara, yaitu:

a. Menggunakan object RegExp. Dimana item pertama dari pola yang cocok diwakili dengan simbol `$1`, item kedua dengan `$2` dan seterusnya.

```
re.exec(date);
RegExp.$1;
RegExp.$2;
RegExp.$3;
```

b. Menyimpan item hasil pencocokan pola didalam sebuah variabel array.

```
var result = re.exec(date);
result[1];
result[2];
result[3];
```

2.2.3 Metode Replace

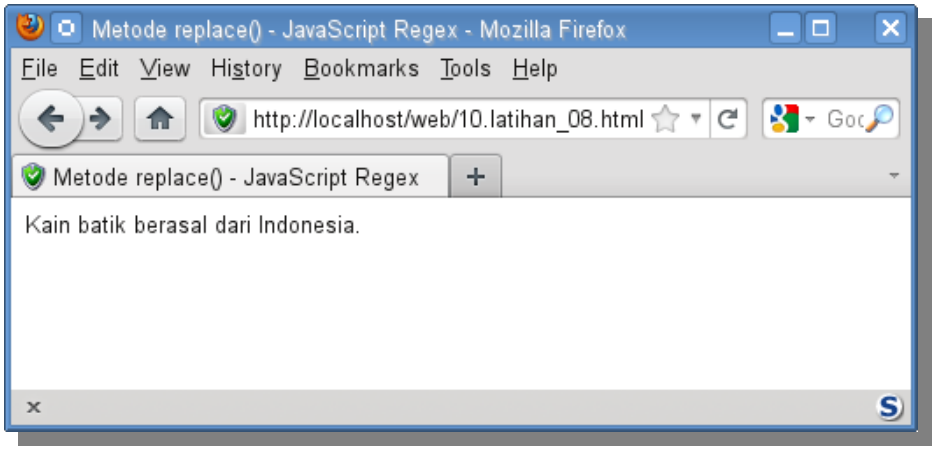
Fungsi ini digunakan untuk mengganti string yang cocok dari pola regex dengan nilai yang telah ditentukan. Contoh :

10.latihan_08.php

```
<html>
   <title>Metode Replace - JavaScript Regex</title>
   <body>
      <script language="JavaScript">
         var text    = "Kain batik berasal dari Malaysia.";
         var pattern = /Malaysia/;

         text = text.replace(pattern, "Indonesia")

         document.write(text);
      </script>
   </body>
</html>
```



3. Optimasi

Terdapat dua elemen yang terlibat dalam operasi regex. Pertama adalah polanya dari regex itu sendiri, yaitu sebuah string yang berisi deretan karakter, karakter meta, dan sintaks regex yang lain. Kedua adalah *mesin regex*, yaitu komponen yang ada di bahasa pemrograman atau aplikasi yang akan memroses pola tersebut.

Mesin regex mula-mula akan membaca pola, lalu *mengkompilasinya* menjadi bentuk internal graf atau seperti peta berpanah. Misalnya, pola `a|b` akan diubah menjadi semacam bentuk bercabang dua. Selanjutnya, mesin regex akan mengambil string lalu memasukkannya ke dalam graph yang telah dibuat. Mesin regex akan berusaha melewatkan string ini melalui cabang-cabang yang sesuai dari titik masuk graf hingga titik akhir.

Jika berhasil mencapai finish, maka string tersebut dikatakan cocok dengan pola. Jika gagal atau buntu, maka string dikatakan tidak cocok dengan pola. Mesin regex akan mencoba setiap cabang yang mungkin, bergerak mundur lalu mencoba cabang lainnya bila perlu, sampai tidak ada alternatif lagi yang memungkinkan. Karena itu semakin rumit

dan panjang pola Anda, kemungkinannya adalah proses kompilasi dan pencocokan akan semakin lama. Oleh karena itu hendaklah menuliskan pola regex sesuai dengan kebutuhan saja.

Sebagaimana jika mesin regex menemukan pola seperti `.+` atau `.*`, mesin regex akan langsung melahap sisa string sebanyak-banyaknya dalam pencocokan, hingga akhir baris atau bahkan kalau perlu hingga akhir teks (bergantung pada kehadiran modifier `s`). Perhatikan contoh program PHP berikut ini :

```
$text = "Aku seorang kapitan. Mempunyai pedang panjang.";
$text = preg_replace("/s.+g/","",$text);
```

Jika kita perhatikan, skrip program diatas digunakan untuk menghilangkan string yang diawali dengan huruf `s` dan diakhiri huruf `g`. Jika anda mengira bahwa program diatas hanya akan menghapus string `seorang`, maka anda keliru.

```
"Aku seorang kapitan. Mempunyai pedang panjang."
```

Perhatikan huruf **s** dan **g** yang dicetak tebal diatas? Mesin regex akan menghapus seluruh string diantara huruf **s** dan **g** tersebut.

```
"Aku seorang kapitan. Mempunyai pedang panjang."
```

Sehingga fungsi `preg_replace` akan mengembalikan nilai sebagai berikut :

```
"Aku ."
```

Hal ini dikarenakan karakter meta pengulangan `+` dan `*` mempunyai sifat yang **rakus**. Artinya, dia akan mencari dan mencocokkan dengan sebanyak-banyaknya karakter. Pada banyak kasus, biasaya kita lebih membutuhkan pola regex dengan sifat tidak rakus. Sifat rakus ini dapat dihilangkan dengan cara menambahkan karakter meta optional `?` setelah karakter meta pengulangan `+` atau `*`.

Pada kasus ini, kita perlu menambahkan karakter meta optional `?` setelah karakter meta pengulangan `+`, untuk mengubah perilaku mesin regex. Dimana setelah mesin regex menemukan huruf **g** yang pertama, mesin akan berhenti mencari huruf **g** selanjutnya.

Jadi, jika kita hanya ingin menghilangkan string `seorang` dari variabel `$text`, kita dapat menulis ulang program PHP diatas menjadi seperti ini dibawah :

```
$text = "Aku seorang kapitan. Mempunyai pedang panjang.";
$text = preg_replace("/s.+?g/","",$text);
```

String yang akan dibuang oleh program diatas adalah seperti yang dicetak tebal berikut :

```
"Aku seorang kapitan. Mempunyai pedang panjang."
```

Sehingga string yang akan dikembalikan oleh fungsi `preg_replace` dan disimpan kedalam variabel `$text` adalah sebagai berikut :

```
"Aku kapitan. Mempunyai pedang panjang."
```

4. Contoh-Contoh : Validasi Email Sederhana

10.latihan_09.php

```
<html>
   <head>
      <title>Simple PHP Email Validation</title>
   </head>
   <body>
      <h2>Pengecekan Format Email Sederhana</h2>
      <div id="txtStatus">
      <?php

      if(isset($_POST['btnSubmit'])){
         if(preg_match("/[a-z0-9\._]+?@[a-z-\.]+\.[a-z]{2,6}$/",
            $_POST['txtEmail'])){
            echo "Format email valid";
         } else {
            echo "Format email tidak valid";
         }
      }

      ?>
      </div>
      <form action="" method="post">
         Email Anda : <input type="text" name="txtEmail" id="txtEmail" />
         <br />
         <input type="submit" name="btnSubmit" value="Periksa" />
      </form>
   </body>
</html>
```

10.latihan_10.html

```
<html>
   <head>
      <title>Simple JavaScript Email Validation</title>
      <script language="JavaScript">
         function formValidation(){
            var rgx = /[a-z0-9\._]+?@[a-z-\.]+\.[a-z]{2,6}$/;

            if(rgx.test(document.getElementById("txtEmail").value)){
               str = "Format email valid";
            } else {
               str = "Format email tidak valid";
            }
```

```
            document.getElementById("txtStatus").innerHTML = str;
            return false;
         }
      </script>
   </head>
   <body>
      <h2>Pengecekan Format Email Sederhana</h2>
      <div id="txtStatus"></div>
      <form action="" method="post" onSubmit="return formValidation();">
         Email Anda : <input type="text" name="txtEmail" id="txtEmail" />
         <br />
         <input type="submit" name="btnSubmit" value="Periksa" />
      </form>
   </body>
</html>
```

DAFTAR PUSTAKA

1. Forta, Ben. 2004. *Teach Yourself Regular Expression in 10 Minutes*. Sams Publishing.
2. Goyvaerts, Jan., Steven Levithan. 2009. *Regular Expression Cookbook*. O'Reilly Media, Inc.
3. Haryanto, Steven. *Kuasai Regex Hari Ini Juga! - Web Master Magazine*. http://www.master.web.id/mwmag/issue/. Diakses 05 Desember 2011.
4. *Regular Expression Cheat Sheet*. http://www.addedbytes.com/cheat-sheets/regular-expressions-cheat-sheet/. Diakses 10 Desember 2011.
5. Stubblebine, Tony. 2003. *Regular Expression Pocket Reference*. O'Reilly Media, Inc.
6. Rasmus, Lerdorf., Dkk. 2003. *PHP 5 Manual*. PHP Documentation Group.