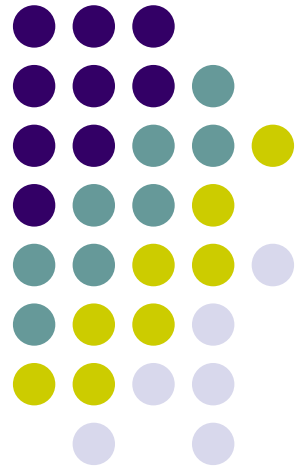
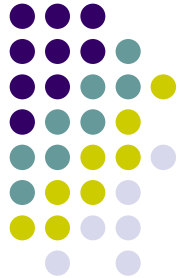
Teknik Kompiler 4

oleh: **antonius rachmat c, s.kom, m.cs**



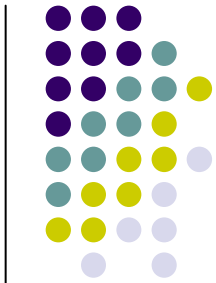
REGULAR EXPRESSION 2
Karakter Meta (lanj.)



- **Karakter whitespace**
  - Menggunakan sintaks **\s** atau **\S** sebagai komplemennya.
  - Yang termasuk whitespace adalah spasi, tabulasi, dan newline. Contoh : ^\S berarti mencari baris yang tidak diawali dengan whitespace, atau baris yang tidak kosong atau diindentasi.
  - Bagaimana mencocokan string ini ? “aku mau makan nasi”, “akumaumakannasi”?

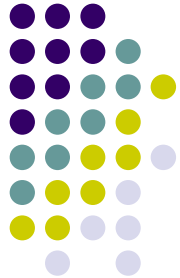
- **Batas antar kata (word boundary)**
  - \b berarti posisi batas antar kata.
  - Akhir dan awal string merupakan batas antar kata.
  - Contoh : “^saya\b” berarti akan cocok hanya dengan baris yang diawali dengan kata “saya”, tapi tidak dengan “sayang”, “sayatan”.

Karakter Meta Dasar



- **Awal dan Akhir String/Baris**
  - Selain menggunakan ^ dan $ maka dapat menggunakan karakter meta \A, \Z, \z.
  - Karakter \A selalu cocok dengan **awal string**.
  - \Z akan selalu cocok dengan **akhir string** atau **sebelum new line**.
  - Sedangkan \z akan cocok dengan **karakter terakhir** dari string.
  - Contoh untuk string “kuku ku\nkakiku\nkaku\n”
  - Yang cocok:
    - \Akuku, \Akuku ku, kaku\Z
  - Tidak cocok:
    - \Akakiku, kakiku\Z, kaku\z, kakiku\z
    - Kenapa?

Karakter Meta Dasar

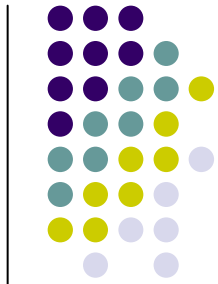


- **Mengescape Karakter Meta**
  - Jika karakter-karakter seperti *,.,),(,[,] dan lain-lain adalah karakter meta yang memiliki arti khusus, lalu bagaimana membuat pola untuk menangani karakter-karakter diatas?
  - Kita harus mengescapenya!
  - Dengan cara memberi karakter \ (backslash). Contoh \(.+\.\). Atau contoh lain : [a-z]\^d+ yang akan mencoccokan string seperti a^3, b^2 dan seterusnya.
  - Bagaimana untuk mengecek bilangan floating point seperti misalnya 123.34? atau 400 atau 0.5?

Escape Karakter Kontrol

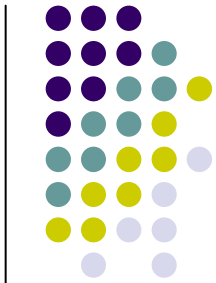
Binary	Oct	Dec	Hex	Abbr	PR[a]	CS[b]	CEC[c]	Description
000 0000	000	0	00	NUL	␀	^@	\0	Null character
000 0001	001	1	01	SOH	␁	^A		Start of Header
000 0010	002	2	02	STX	␂	^B		Start of Text
000 0011	003	3	03	ETX	␃	^C		End of Text
000 0100	004	4	04	EOT	␄	^D		End of Transmission
000 0101	005	5	05	ENQ	␅	^E		Enquiry
000 0110	006	6	06	ACK	␆	^F		Acknowledgment
000 0111	007	7	07	BEL	␇	^G	\a	Bell
000 1000	010	8	08	BS	␈	^H	\b	Backspace[d][i]
000 1001	011	9	09	HT	␉	^I	\t	Horizontal Tab
000 1010	012	10	0A	LF	␊	^J	\n	Line feed
000 1011	013	11	0B	VT	␋	^K	\v	Vertical Tab
000 1100	014	12	0C	FF	␌	^L	\f	Form feed
000 1101	015	13	0D	CR	␍	^M	\r	Carriage return[h]

Karakter Meta Dasar



- **Escape Pola**
  - Jika kita menggunakan banyak sekali escape karakter, tentu akan sangat menjengkelkan menuliskan backslash-backslash terus menerus.
  - Di **beberapa** bahasa pemrograman RE disediakan \Q … \E yang berguna mengapit subpola yang dijadikan literal.
  - Contoh : …* akan mencari 2 karakter apa saja, karakter yang terakhir boleh ada atau tidak (dari 0 sampai tak terhingga). Sedangkan \Q…*\E akan mencari string "…*" sebagaimana mestinya.

Bentuk-bentuk modifier



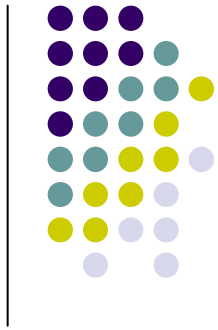
Modifier mampu mengubah kelakuan suatu karakter meta dan literal tertentu.

- *Modifier* i (*case in-case-sensitive*)

  Defaultnya RE akan menggunakan mode case-sensitive yaitu membedakan huruf besar dan kecil. Mode ini agak lebih lambat daripada mode biasanya. Dengan menggunakan mode ini maka [a-z] akan sama dengan [a-zA-Z]. Contoh penggunaan di PHP misalnya:

  ```
  preg_match("/anton/i","saya Anton namanya");
  ```

Modifier (2)

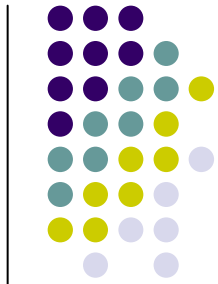


- **Modifier s (single line)**

  Defaultnya: meta . (titik) akan cocok dengan karakter apa saja kecuali \n (newline).
- Jika menggunakan mode ini maka karakter meta . (titik) akan benar-benar cocok dengan karakter apa saja termasuk \n.

  Contoh: `preg_match("/a.ton/s","a\nton");`
- **Tergantung encodingnya**
  - Unix: \n
  - Windows: \r\n

Modifier (3)



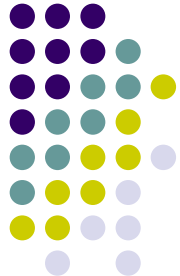
- ***Modifier* m (*multiline*)**
  - Defaultnya ^ dan $ akan berarti awal dan akhir string.
  - Contoh:

    Saya akan membaca buku,

    Kemudian saya akan makan,

    Lalu tidur siang.
  - Maka pola “^Kemudian” tidak akan cocok karena “Kemudian” terletak pada awal baris bukan awal string.
  - Demikian pula dengan “makan,$” tidak akan cocok karena akhir baris, bukan akhir string.
  - Dengan menggunakan mode m maka tanda ^ dan $ akan menjadi awal dan akhir baris.

Modifier (4)



- *Modifier* x (*eXtended legibility*)

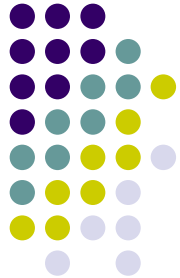
  Regex sulit dibaca karena:

  - Banyak mengandung karakter non angka dan non huruf
  - Banyaknya meta karakter yang harus dihapalkan
  - Amat ringkas sehingga tampak berjejal-an

  Mode ini akan mempermudah penulisan RE karena pola RE akan dapat dipisah menjadi banyak baris dan dapat disertai dengan komentar. Dengan mode ini semua *whitespace* akan diabaikan. Mode ini akan menjadi "wajib" untuk RE yang kompleks.

  Karakter "#" menjadi karakter meta di modifier x

Contoh



- Contoh RE:

  (ftp|http):// # protocol

  ([_a-z\d\-]+(\.[_a-z\d\-]+)+) # TCP addr

  ((/[_a-z\d\-\\\.]+)+)* # unix path

- Contoh inputan :

  Welcome to my homepage http://anso.da.ru.

  E-cash http://www.paycash.ru or http://195.239.62.97/default.htm!

Contoh (2)

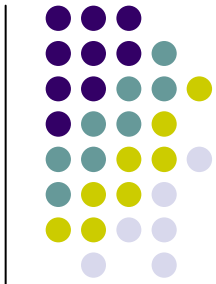
- Contoh RE lain:

```
[+-]?           #cocokkan tanda +/- jika ada
(               #cocokkan mantisa
(\d+\.\d+)      #dalam bentuk a.b
|(\d+\.) #dalam bentuk a.
|(\.\d+) #dalam bentuk .b
|(\d+)           #dalam bentuk bilangan bulat
)
([eE][+-]?\d+)? #cocokkan exponen jika ada
```

- Contoh input : +1.456 atau -1.36e5
- Dalam versi panjang:

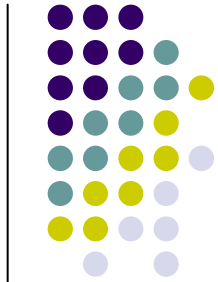
```
[+-]?((\d+\.\d+)|(\d+\.)|(\.\d+|\d+))([eE][+-]?\d+)?
```

Bentuk Modifier lain



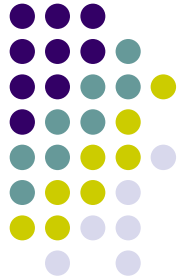
- Sebenarnya masih ada modifier lain yang tidak standar, misalnya:
  - A (*Anchored*), selalu dianggap ada **^**
  - D (*Dollar End Only*), karakter **$** sbg karakter meta jika di akhir string, kalau di tengah dianggap biasa
  - S (*Study*), kompilasi lama, tapi eksekusi cepat
  - U (*Ungreedy*), kebalikan dari greedy
  - E (*Extra*), jika ada \ pada karakter yg tidak dikenal akan dianggap error
  - u (UTF8), dianggap karakter UTF8

Match Group



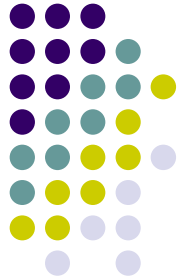
- Untuk mengambil/mengingat subpola dari keseluruhan pola yang terpilih digunakan tanda kurung ( ).
- Subpola itu akan diingat oleh RE dan dapat dipanggil kembali dengan karakter meta **\1, \2**, dan seterusnya untuk setiap subpola yang ditangkap
- **\1, \2, \3**, dan seterusnya itu disebut ***Match Group***.
- Contoh: (aku|kamu)

Contoh Match Group



- Pola 1 : \w+-\w+
  - Yang cocok : kupu-saya
- Pola 2 : (\w+)\s+-\s+\1
  - Yang cocok : kupu-kupu, kata - kata
- Pola 3 : (\w+)-(\w+) \1-\2 \1-\2
  - Yang cocok: do-re do-re do-re
    Pola 1 hanya akan mencocokkan 2 kata yang dipisahkan oleh tanda – entah itu kata yang sama atau bukan.
    Sedangkan pola kedua akan mencocokkan kata ulang!
- Pola : (A\d){2}
  - Yang cocok : A3A1, A1A2

Non Greedy Matching



- Menggunakan sintaks :

  *?, +?, ??, {n}?, {m,n}?, {n,}?
- Artinya hampir mirip dengan sintaks:

  *, + , ?, {n}, {m,n}, dan {n,}
- Untuk mendapatkan pola sesedikit mungkin
- Misalkan string : “mahasiswi informatika ukdw”
  - Pola : maha(.+)i
  - Maka yang didapat adalah : “mahasiswi informati”
  - Hal ini karena RE berkelakukan melahap (*greedy*) seluruh teks hingga huruf i terakhir. Padahal maksud tujuan kita adalah mendapatkan string : “mahasi”, yaitu pencocokkan sesedikit mungkin.
  - Non Greedy akan mampu melakukan itu dengan mengubah pola menjadi: maha(.+?)i

Contoh non-greedy matching

Contoh lain:

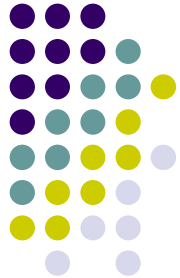
- <!-(.*)--> akan mecocokkan string:

  “HTML ini ada komentar <!- selip --> komentar <!- selip -->”.
- Maka yang didapat adalah :

  “<!- selip --> komentar <!- selip -->”
- Sedangkan \1 yang didapat adalah :

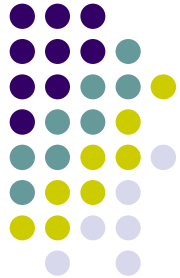
  “selip --> komentar <!- selip”
- Jika ingin mengambil hanya “selip” saja, maka pola RE adalah <!-(*?)-->

Non greedy matching (2)



- Non-greedy hanya cocok untuk mencocokkan pola di bagian tengah saja, bukan pada akhir string.
- Contohnya:
- Pola “komen(.*?)” bermaksud mendapatkan string “komentar” tapi yang kita dapat adalah “komen”. Karena non-greedy akan mencocokkan .*? sebagai 0 atau lebih karakter, dan yang diambil adalah yang paling sedikit, yaitu 0 karakter.
- Sedangkan jika kita ingin mengubah pola menjadi “komen(.+?)”, maka yang didapat hanya “koment” karena .+ akan mencocokkan 1 atau lebih karakter, dan yang diambil 1 karakter saja.

Non Capturing Group

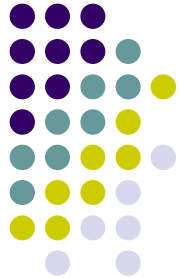


- Sintaks : (?:P), dimana P adalah subpola
- Contoh:

  Pola : ((\(\d{3,4}\))-(\d+))

  String : (0274)-515278
  - Maka \0 akan menghasilkan (0274)-515278,
  - \1 akan menghasilkan (0274)-515278,
  - \2 akan menghasilkan (0274) dan
  - \3 akan menghasilkan 515278
- Jika kita menggunakan non-capturing group, maka pola akan menjadi:
- Pola : (?:(\(\d{3,4}\))-(\d+))
  - Maka \0 akan menghasilkan (0274)-515278,
  - \1 akan menghasilkan (0274) dan
  - \2 akan menghasilkan 515278

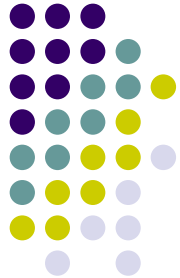
Positive Look



- ***Positive Look Ahead***
  - Sintaks : (?=P) dimana P adalah subpola
  - Pola ini akan mencari pola yang **diikuti** oleh pola tertentu
  - Awalnya : \w+\s*(\d)
    - anton 123
  - Sekarang : \w+\s*(?=\d)
    - anton 123.
  - Awalnya : \w+(\s\d)
    - anton 123
  - Sekarang : \w+(?=\s\d)
    - anton 123

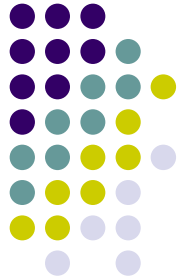
- ***Positive Look Behind***
  - Sintaks : (?<=P)
  - Awal: \d\s*[a-z]+
    - String yang cocok : 123 anton
  - Sekarang: (?<=\d)\s*[a-z]+
    - String yang cocok : 123 anton
  - Pola ini akan mencari pola tertentu yang **didahului** oleh pola tertentu.

Negative Look



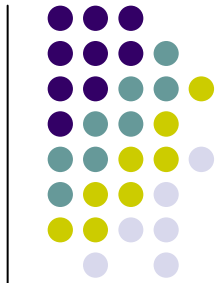
- **Negative look ahead**
  - Untuk mencari string yang **tidak diikuti** oleh pola tertentu.
  - Sintaks : (?!=P)
  - Contoh : \w+(?!=\s\d)
  - Mencari kata yang **tidak diikuti** oleh spasi dan angka
- **Negative look behind**
  - Untuk mencari string yang **tidak didahului** oleh pola tertentu
  - Sintaks : (?<!P)
  - Contoh : (?<!\d\s)\w+
  - Mencari kata yang **tidak didahului** oleh pola tertentu

Named Group



- Python dan .NET mendukung penamaan group pola. Sintaksnya:
- Di .NET : (?<GroupName>Pattern) atau (?'GroupName'Pattern)
- Contoh string : Temperature:22.2
- Contoh pola : Temperature:(?<derajat>\d{2}\.\d)

RegExps in Java

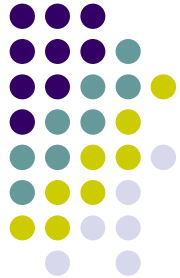


- Two important classes:
  - **java.util.regex.Pattern** -- a compiled representation of a regular expression
  - **java.util.regex.Matcher** -- an engine that performs match operations by interpreting a Pattern
- Example

```
Pattern p = Pattern.compile("a*b");
Matcher m = p.matcher("aaaaab");
boolean b = m.matches();
```

- To produce a slash in a Java String: “//”

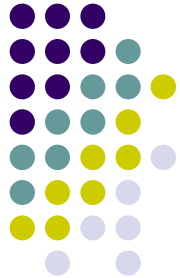
Example



```
import java.util.regex.*;
public class RegEx{
   public static void main( String args[] ){
      String amounts = "$1.57 $316.15 $19.30 $0.30 $0.00 $41.10 $5.1 $.5";
      Pattern strMatch = Pattern.compile( "\\$(\\d+)\\.(\\d\\d)" );
         Matcher m = strMatch.matcher( amounts );
         while ( m.find() ){
            System.out.println( "$" + ( Integer.parseInt( m.group(1) ) + 5 )
              + "." + m.group(2) );
         }
   }
}
```

=> Adds $5 to every amount except the last two

NEXT



- Latihan!
- Teori Bahasa dan Automata tentang Grammar, DFA, dan NFA