Alla scoperta della regular expressions

Flavio <micron> Castelli
<micron@madlab.it>

Introduzione

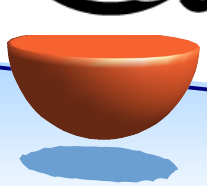
Definizione

Cosa è una *regular expression*?

Una regular expression (o espressione regolare) è una stringa che, tramite un'apposita sintassi, identifica un insieme di stringhe.

Possibili sinonimi:

- regexp (plurale regexps)
- regex (plurale regexes)
- regxp (plurale regexen)



Introduzione

Ambiti d'utilizzo

Quando possiamo utilizzare le regular expression?

- programmando
- da console, in maniera "interattiva" oppure tramite script
- all'interno di normali programmi: editor di testo, viewer,...

In poche parole... **sempre**!

Introduzione

Usi comuni

Le regular expression sono usate principalmente per:

- validare del testo
- ricercare pattern in un testo

Introduzione

Usufruitori

Molte persone credono che le regexp siano uno strumento riservato a un'elite.

FALSO!

Con una base di conoscenze è possibile avere notevoli vantaggi, anche nella vita di tutti i giorni.

Introduzione

Il primo esempio

Questa è una semplicissima regexp:

```
BgLug
```

Applichiamola alle seguenti stringhe:

Il Lug di Bergamo (Bg) si chiama BgLug

MATCH

Il Lug di Bergamo (Bg) si chiama BGLug

NESSUN MATCH

Prima regola: le regexp sono case sensitive

La sintassi

Metacaratteri

Le regexp sono definite da una sequenza di caratteri.

Non tutti i caratteri sono interpretati in modo letterale.

→ **Metacaratteri**

Caratteri con una valenza particolare

Usati per espandere le potenzialità delle regexp

La sintassi

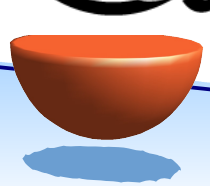
Metacaratteri - simbolo '.'

Il simbolo '.' può assumere qualsiasi valore (numero, lettera, simbolo)*

E' una sorta di carattere "jolly"

Esempio:
La regexp `B.Lug` può assumere i seguenti valori:

- BoLug
- BOLug
- BàLug
- B4Lug
- B!Lug
- ... e molti altri ancora ...



* fatta eccezione per il carattere newline ('\n')

La sintassi

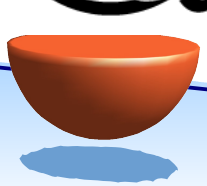
Metacaratteri - simbolo '*'

Il simbolo '*' indica una ripetizione del carattere precedente

Il carattere si può ripetere n volte, con n ≥ 0

Esempio:
La regexp `B*Lug` può assumere i seguenti valori:

- Lug
- BBLug
- BBBLug
- ... e molti altri ancora ...



La sintassi

Metacaratteri - simbolo '+'

Il simbolo '+' indica una ripetizione del carattere precedente

Il carattere si può ripetere n volte, con n > 0

Esempio:
La regexp `B+Lug` può assumere i seguenti valori:

- BLug
- BBLug
- BBBLug
- ... e molti altri ancora ...

La sintassi

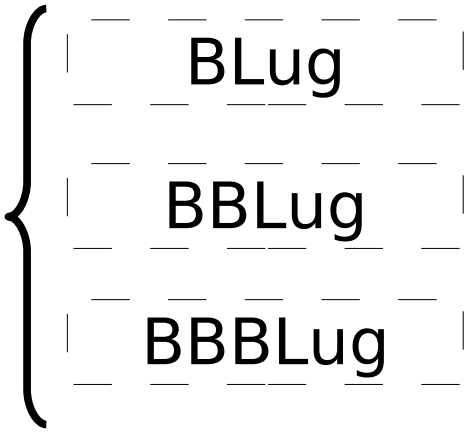
Quantificatori

Per definire in modo preciso il numero di ripetizioni si usa la sintassi: $\{\alpha,\beta\}$

Questo significa che il carattere precedente a '{' si ripete n volte, con $\alpha \leq n \leq \beta$
NB: se β non è specificato vale ∞

Esempio:

B{1,3}Lug



ABBREVIAZIONE: il simbolo '**?**' è equivalente alla sintassi {0,1}

La sintassi

Raggruppamento

E' possibile raggruppare dei pattern usando '**(**' e '**)**'

`(pattern)`

`(Bg)*Lug` { Lug, BgBgBgLug, ⋮

`(Bg)+Lug` { BgLug, BgBgLug, ⋮

La sintassi

Alternative

Il simbolo '|' indica un'alternativa tra due pattern.

```
pattern1 | pattern2
```

E' come il predicato logico *or*.

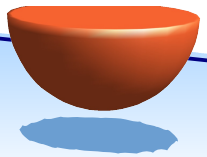
```
(Bg)|(Bo)Lug
```

```
BgLug
```

```
BoLug
```

Una sintassi alternativa potrebbe essere:

```
B(g|o)Lug
```



La sintassi

Range

Per fare il match su un range di parametri si usano '[' e ']'

Esempi:

`[begin – end]`

[0-9]	tutti i numeri tra 0 e 9
[a-g]	tutte le lettere minuscole tra 'a' e 'g'
[a-df-z]	tutte le lettere minuscole tranne 'e'
[a-zA-Z]	tutte le lettere minuscole e maiuscole
linux day 200[0-2]	• linux day 2000 • linux day 2001 • linux day 2002

La sintassi

Range - abbreviazioni

Esistono delle abbreviazioni per alcuni range di caratteri

[\d]	un numero
[\D]	tutto tranne un numero
[\w]	una lettera o un numero
[\W]	ne una lettera ne un numero
[\s]	uno spazio
[\S]	tutto tranne uno spazio

La sintassi

Ancore - definizione

E' possibile “ancorare” le posizione di un pattern.

Esistono i seguenti tipi di ancore:

- all'**inizio** della stringa: `^(pattern)`
- alla **fine** della stringa: `(pattern)$`
- di delimitazione: `\b(pattern)`

  `(pattern)\b`

La sintassi

Ancore – esempio 1

Data la regexp:

```
^(BgLug)
```

Il lug di Bergamo si chiama BgLug

NESSUN MATCH

BgLug rulez!

MATCH

Data la regexp:

```
(BgLug)$
```

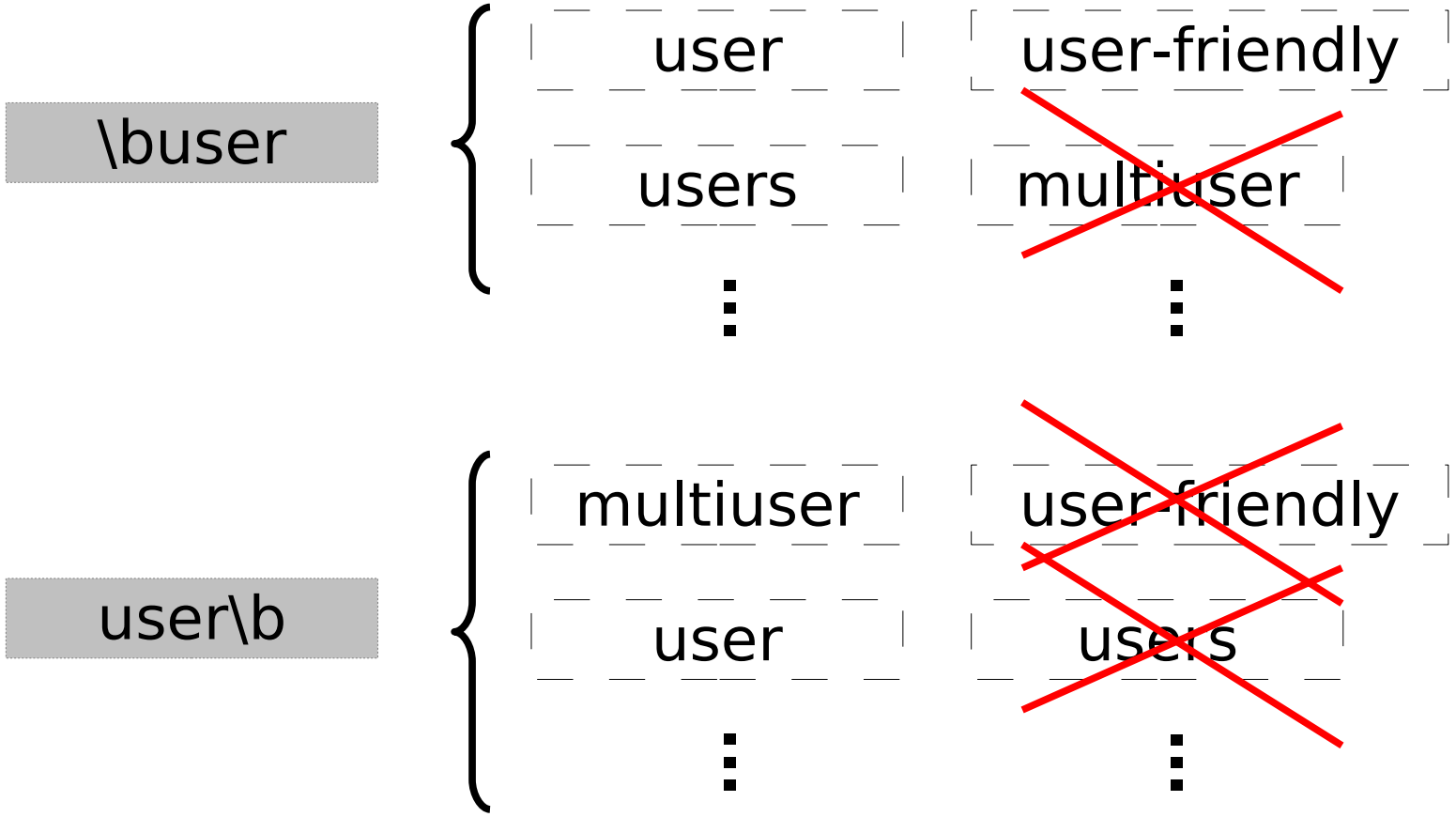
Il lug di Bergamo si chiama BgLug

MATCH

La sintassi

Ancore – esempio 2

Impariamo ad usare il delimitatore **\b**



Implementazione

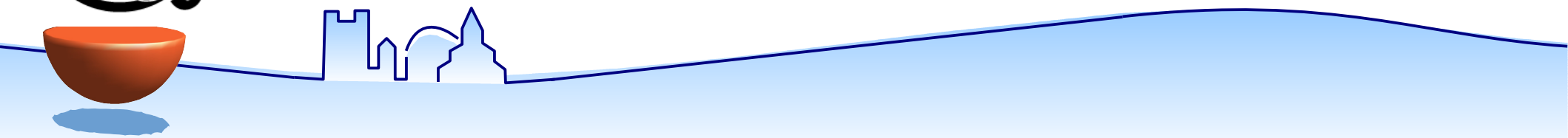
Usiamo le regexp...

Le regular expressions sono disponibili in quasi tutti i linguaggi:

- in modo nativo
- tramite librerie,moduli,... più o meno ufficiali

Saranno mostrate le implementazioni in:

- bash
- c++
- perl
- python



Implementazione - Perl

Dichiarazione

- Supporto nativo (nessuna libreria esterna,...)
- Per dichiarare una regexp si usa la seguente sintassi:

```
/pattern/
```

Esempi:

```
/B(g|o)Lug/
```

```
/\w+/
```

```
/http:\/\/bglug.*/
```

escaping del carattere '/' tramite il simbolo '\'

Implementazione - Perl

Matching

Per verificare la presenza di un pattern in una stringa:

Match su variabile precisa

```
if ($var =~ /pattern/)
    {print “match\n”;}
else
    {print “no match\n”;}
```

Match su $_

```
if (/pattern/)
    {print “match\n”;}
else
    {print “no match\n”;}
```

Implementazione - Perl

Matching - opzioni

E' possibile specificare dei parametri opzionali per il matching:

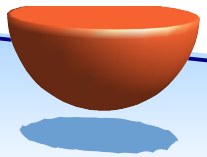
- '**i**': ricerca case-insesitive
- '**s**': considera i new-line come dei simboli

```
$_ = “W BGLug”;
if (/.*bglug/i)
    { print “match\n”;}
```

MATCH

```
$_ = “I love\nBGLug”;
if (/\blove\b.*\bbglug\b/is)
    { print “match\n”;}
```

MATCH



Implementazione - Perl

Sostituzioni

E' possibile sostituire una stringa che fa il match di un pattern

Substitute su variabile precisa

```
$var =~ s/pattern/new_string/;
```

Substitute su $_

```
s/pattern/new_string/;
```

E' possibile usare un qualsiasi delimitatore invece di '/'

```
s#pattern#new_string#/;
```

Implementazione - Perl

Sostituzioni - opzioni

Sono disponibili dei parametri opzionali per il substitute:

- Sono ancora valide le opzioni '**i**' e '**s**' viste precedentemente
- Nuova opzione: '**g**' permette di sostituire tutte le occorrenze del pattern (invece di fermarsi alla prima)

```
$_ = "Questo linus-day è stato organizzato dal
                BOLug,Linus è un grande OS";
s/linus/linux/ig;
s#b.lug#BgLug#i;
print $_,"\n";
```

Questo linux-day è stato organizzato dal
BgLug,linux è un grande OS

Implementazione - Perl

Split

E' possibile dividere una stringa in varie parti, usando come separatore un patter

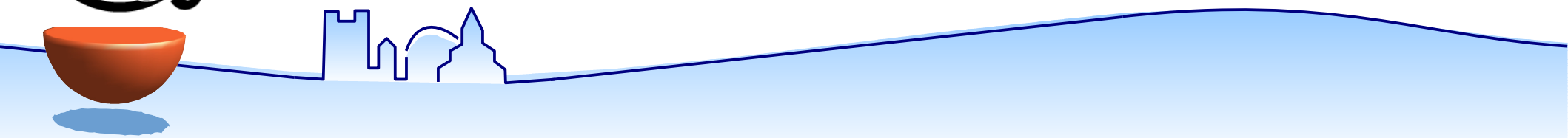
Split su variabile precisa

```
@splitted = split (/pattern/, $var);
```

Split su $_

```
@splitted = split /pattern/;
```

'**split**' restituisce le varie sotto-stringhe all'interno di un array

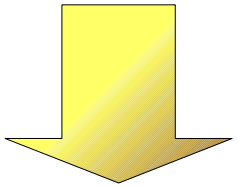


Implementazione - Perl

Split -emempio

Separiamo i campi di un time stamp:

```
$_ = "25/11/2005 17:30";
@splitted = split (/\D/,$_);
foreach (@splitted)
      {print "|$_|\n";}
```

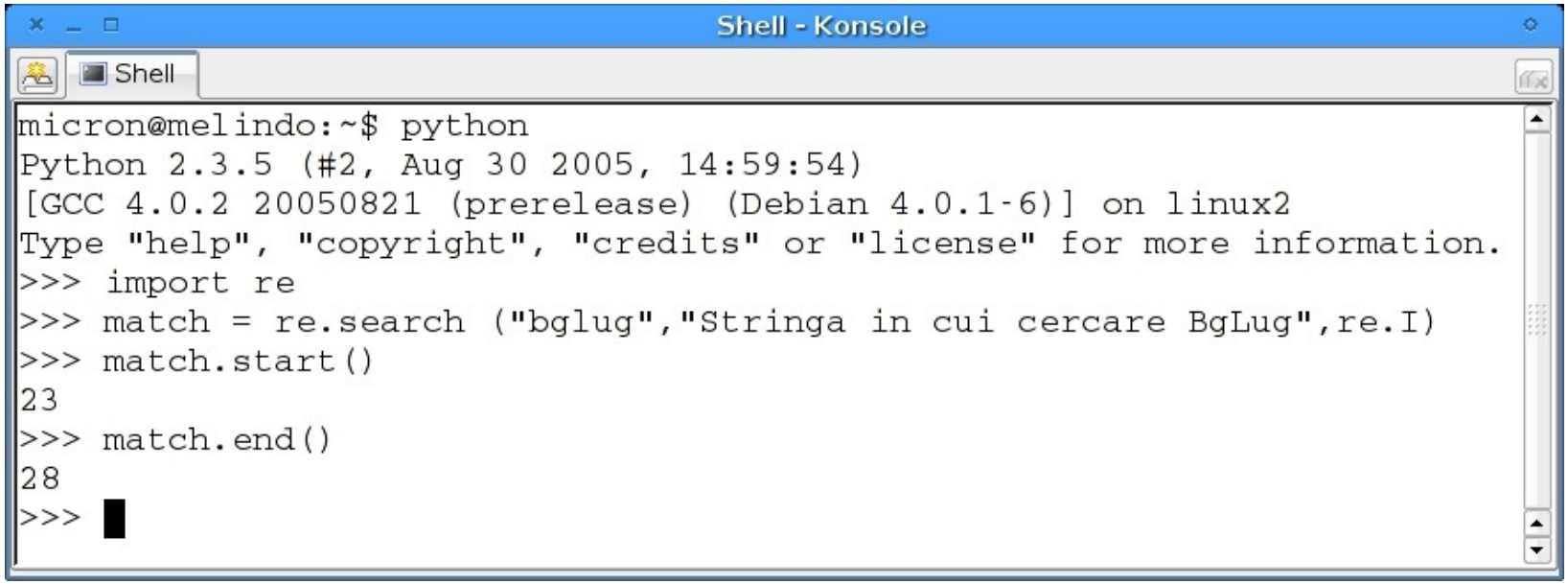


```
|25|
|11|
|2005|
|17|
|30|
```

Implementazione - Python

Supporto

- Supporto non nativo
- Sono fornite dal modulo '**re**' → `import re`
- E' possibile sfruttare la console di python per delle prove “al volo”



Implementazione - Python

Search - esempio

```
import re
pattern = re.compile (“bglug”)
string = “I love BgLug”
```

1)import delle modulo per le regexp
2)creiamo il pattern
3)creiamo una stringa

```
match = re.search (string, pattern,re.I)
match.start()
```

1)cerchiamo il pattern nella stringa
2)la prima occorrenza del pattern

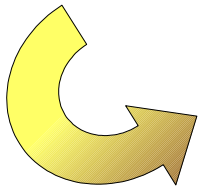
Implementazione - Python

Substitute

```
import re
pattern = re.compile (“windows”)
str_replace = “linux”
string = “I like windows”
```

1)creiamo il pattern
2)creiamo la stringa per le sostituzioni
3)creiamo una stringa

```
re.sub (pattern, str_replace, string)
```



I lile linux

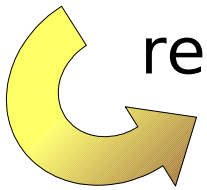
Implementazione - Python

Split

```
import re
pattern = re.compile (“\D”)
string = “26/11/2005 17:00”
```

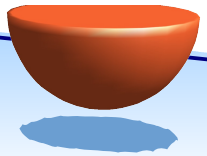
1)creiamo il pattern
2)creiamo una stringa

```
re.split (pattern, string)
```



restituisce un array con i token

['26', '11', '2005', '17', '00']



Implementazione - Python

Search

- Dichiarazione del pattern:

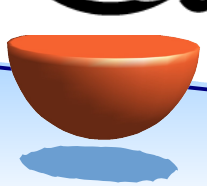
```
pattern = re.compile (“..Lug”)
```

- Search su una stringa:

```
match = re.search (stringa, pattern,flags)
```

restituisce un'istanza di tipo **MatchObject** o **None**

- **re.I**: search case-insesitive
- **re.S**: '.' può assumere anche valore di '\n'
- ...



Implementazione - C++

Supporto

- Supporto non nativo
- Sono fornite da varie librerie:
  - **boost** [http://www.boost.org]
  - **Qt** [http://www.trolltech.com/]
  - **wxWidgets** [http://www.wxwidgets.org/]
  - ...

Pro:
- installazione rapida

Contro:
- sviluppando una GUI è un po' scomodo usarla

http://www.boost.org/libs/regex/doc/

Implementazione - C++

Search – parte I

header libreria boost

```
#include <boost/regex.hpp>
#include <string>

int main()
{
        boost::regex pattern ("bg|olug",boost::regex_constants::icase |
boost::regex_constants::perl);
        std::string stringa ("Searching for BsLug");
```

regexp

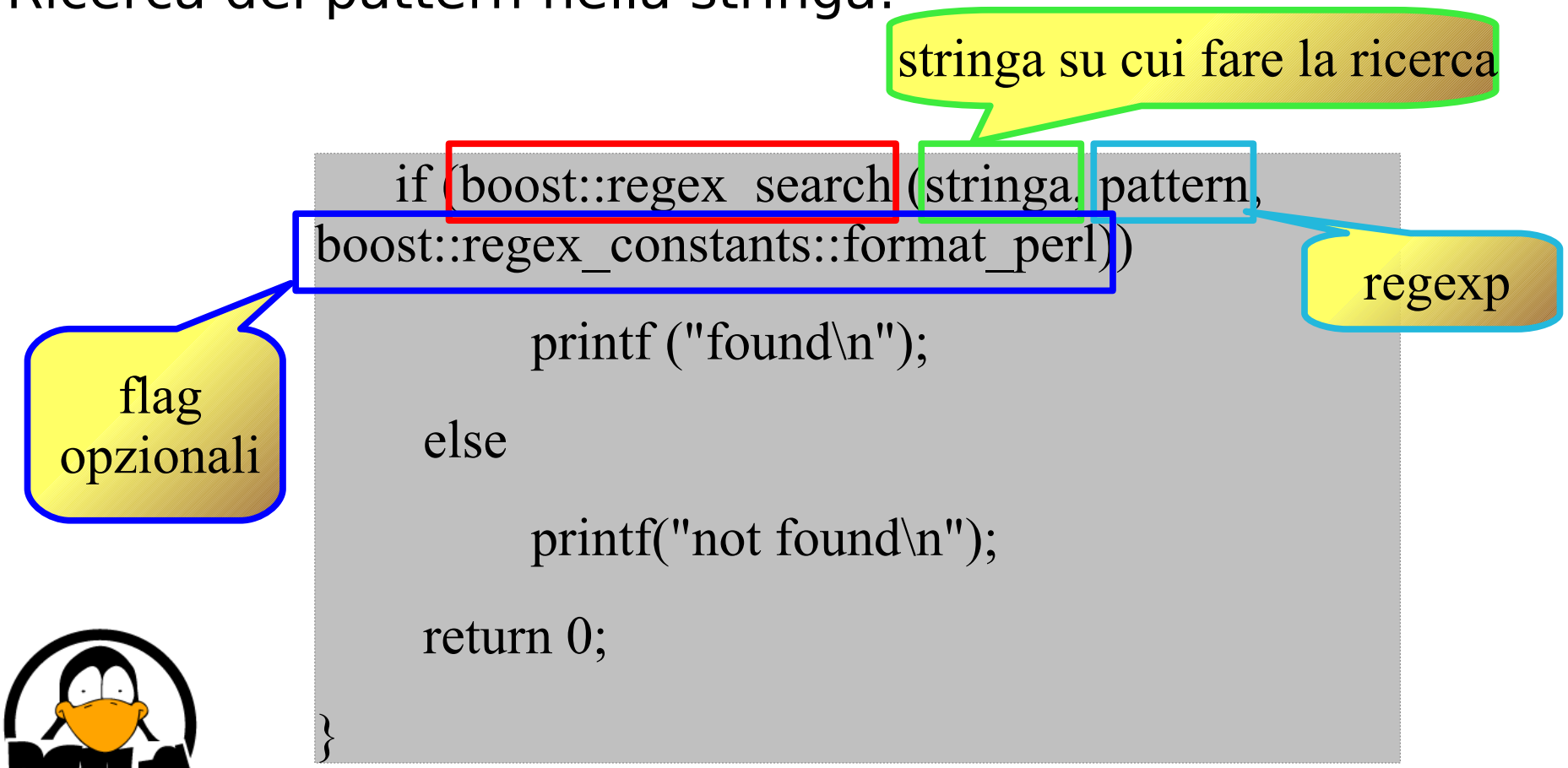
flag opzionali

Dichiarazione del pattern

Implementazione - C++

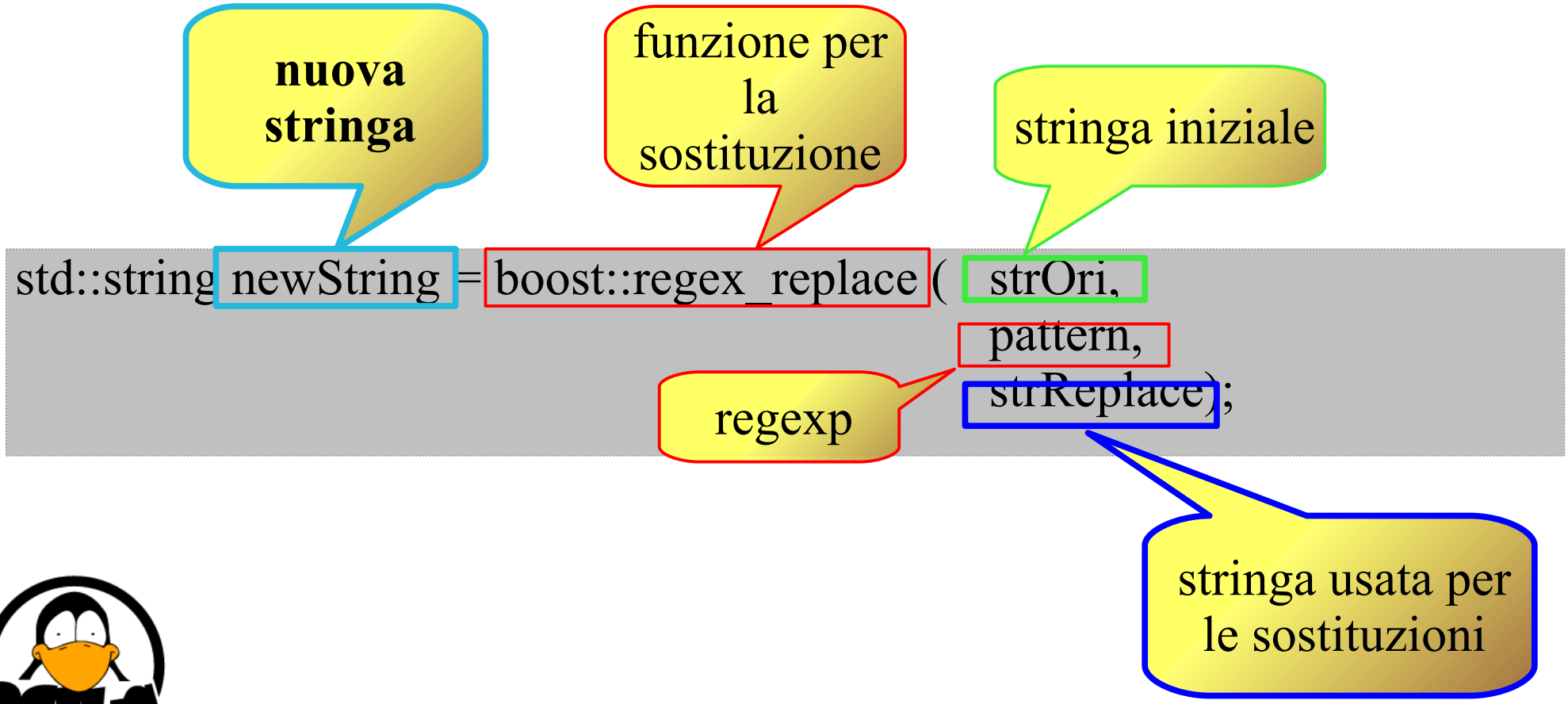
Search – parte II

Ricerca del pattern nella stringa:



Implementazione - C++

Replace



Implementazione - C++

Split – parte I

```
#include <boost/regex.hpp>
#include <string>
int main()
{
        boost::regex pattern ("\\D",boost::regex_constants::icase|
boost::regex_constants::perl);
        std::string stringa ("26/11/2005 17:30");
        std::string temp;
        boost::sregex_token_iterator i(stringa.begin(), stringa.end(), pattern,
-1);
        boost::sregex_token_iterator j;
```

regexp

iteratore sulla stringa, punta alle occorrenze del pattern

iteratore di fine sequenza

Implementazione - C++

Split – parte II

```
    unsigned int counter = 0;
    while(i != j)
    {
        temp = *i;
        printf ("token %i = |%s|\n", ++counter, temp.c_str());
        i++;
    }
    return 0;
}
```

ciclo per tutte le occorrenze del pattern

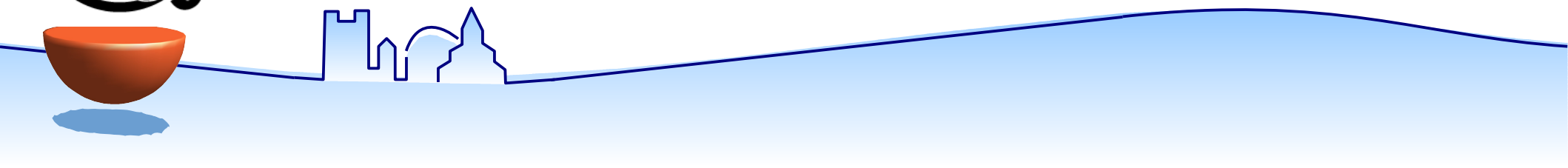
'**i**' punta alla stringa ottenuta tramite lo split

```
token 1 = |26|
token 2 = |11|
token 3 = |2005|
token 4 = |17|
token 5 = |30|
```

Implementazione - Bash

Supporto

- Supporto non nativo delle regexp
- Esistono una serie di programmi che le implementano:
  - sad
  - awk
  - grep
  - ...
- Questi programmi possono essere usati:
  - all'interno di script (programmazione)
  - da linea di comando



Implementazione - Bash

Grep

- E' lo strumento per eccellenza per eseguire ricerche
- Come usare le regexp con grep:

```
grep 'pattern' file1 file2 ...
```

Attenzione: è possibile racchiudere la regexp tra apici singoli (') oppure doppi (“).

E' consigliato usare gli apici singoli, a meno che non si vogliano usare delle variabili definite in bash.

Per esempio:

```
grep “.*$HOME.*” pippo
```

```
grep “.*/home/micron.*” pippo
```

Implementazione - Bash

Grep – esempio di ricerca

Supponiamo di volere trovare tutte le occorrenze del termine “Ati“ nei file di documentazione del kernel

```
micron@melindo:$ cd /usr/src/linux/Documentation
micron@melindo:$ grep -r -i '\bAti\b' *
```

ricerca su tutti i file

Le opzioni di grep:
- '**r**': ricerca ricorsiva
- '**i**': ricerca case insensitive

I '**\b**' servono a eliminare tutti i “falsi positivi” come *'configur**ati**on'*, *'implement**ati**on'*...

Implementazione - Bash

Less

- Permette di visualizzare con comodità dei file di testo
- E' usato per leggere le man pages

Come effettuare una ricerca in less:

1) digitare il simbolo '**/**'
2) scrivere la regexp/termine dopo il '/' e premere invio
3) premere il tasto '**n**' per trovare la prossima occorrenza
4) premere il tasto '**N**' per trovare l'occorrenza precedente

Implementazione - Bash

Less – ricerca case insesitive

Per effettuare una ricerca case insensitive:

- si invoca less con l'opzione '**-i**' o '**-I**'
- prima di invocarlo:

```
micron@melindo:$ LESS="-i"; export LESS
micron@melindo:$ man bash
```

Questo secondo metodo è molto utile quando si usa il comando '**man**'

ATTENZIONE: le opzioni 'i' e 'I' hanno significati diversi:

- '**i**': ricerca case insesitive solo se il pattern non contiene lettere maiuscole
- '**I**': ricerca sempre case insesitive

Alla scoperta della regular expressions

Domande ?