

テキスト処理 第1回 (2008-04-15)

田中哲

産業技術総合研究所

情報技術研究部門

akr@isc.senshu-u.ac.jp

<http://staff.aist.go.jp/tanaka-akira/textprocess-2008/>

今日の内容

- この授業の概要
- Ruby の使いかた
- 正規表現の使いかた
- Rubyのインストール

この授業の概要

- 質問について
- 授業の資料
- 授業の狙い
- テキスト処理とは？
- 授業の構成
- 評価
- 参考書

質問について

- 授業中は講義中いつでも割り込んでよい
- 大学に常駐していないので口頭での質問は授業の後のみ
- メールでの宛先:
`akr@isc.senshu-u.ac.jp`

授業の資料

- なるべく前日までにプレゼン資料を用意する
<http://staff.aist.go.jp/tanaka-akira/textprocess-2008/>

授業の狙い

- 日常生活でテキストを処理するプログラムを書く力を身につける
- テキスト処理の仕組みを理解する
- とくに正規表現の機構と使用法
- 形式言語理論の基礎を学ぶ

テキスト処理とは？

- テキストを処理する (文字どおり)
 - テキスト
 - プレインテキスト
 - HTML
 - XML
 - etc.
 - 処理
 - 検索
 - 集計
 - etc.

授業の構成

- Rubyを使ってみる
- 正規表現を使ってみる
- 正規表現エンジンを作ってみる
- エンジンを作るのに必要な技術を学ぶ
- 正規表現の理論を学ぶ
- 理論にあわないところを学ぶ

評価

- レポートで評価する
基本的に毎回レポートを出す
(履修登録が終わって RENANDI が使えるようになったら)
内容は Ruby を使ってプログラムを書くものになる
- 出席はとらない

参考文献 (Ruby)

- 必要というわけではないが、興味があれば

- たのしいRuby 第2版

<http://www.network.org/sbcr-ruby/>



- Rubyプログラミング基礎講座

<http://www.gihyo.co.jp/books/4-7741-2645-4>

- Rubyリファレンスマニュアル (web)

<http://www.ruby-lang.org/ja/man/>

参考文献 (正規表現)

- 必要というわけではないが、興味があれば
- 詳説 正規表現 第2版
<http://www.oreilly.co.jp/books/4873111307/>
- オートマトン 言語理論 計算論 I [第2版]
http://www.saiensu.co.jp/?page=book_details&SBN=ISBN978-4-7819-1026-0&YEAR=2003



Rubyの使いかた

- プログラミング言語Ruby
- Rubyの起動
- データの表示

プログラミング言語Ruby

- オブジェクト指向スクリプト言語
- 正規表現をサポートしている
- 開発元: <http://www.ruby-lang.org/>
- 自由なライセンス
 - どんな用途にも自由に使っていい
 - 自由に中身を調べて改造してもいい
 - 他のひとに自由に配っていい
 - 改造したものを配るのも自由

Hello World

- Hello World と表示するプログラム
- `print "Hello World\n"`

Hello World の実行

- プログラム: `print "Hello World¥n"`
- 引数から実行
- ファイルから実行
- irbから実行
- C言語などと異なりコンパイルは不要

引数から実行

- コマンドプロンプトから ruby コマンドを起動
- ruby コマンドの引数として -e program と書く
- -e はプログラムを直接指定するオプション

```
% ruby -e 'print "Hello World\n"'  
Hello World
```

- シェルの特殊文字を記述するには細工が必要
(上記ではシングルクォートはそのまま書けない)

ファイルから実行

- ファイルにプログラムを書く
- 拡張子は rb にする (必須ではない)
- ファイル名を引数にして ruby コマンドを起動

```
% cat hello.rb ← ファイルの内容を表示  
print "Hello World\n"  
% ruby hello.rb ← ruby を実行  
Hello World
```

irbから実行

- irb は ruby の対話インターフェース
- コマンドプロンプトから irb を実行する
- irb のプロンプトにプログラムを入力する

```
% irb
```

```
irb(main):001:0> print "Hello World\n"
```

```
Hello World ← 表示結果 (print の副作用)
```

```
=> nil ← print が返した値
```

Rubyの実行

- 引数から実行
% ruby -e 'コード'
- ファイルから実行
% ruby コードの入ったファイル名
- irb (対話型Ruby) から実行
% irb
irb(main):001:0> コード
=> コードの返値

表示と返値

- "Hello World¥n" は文字列の値を返す
- print は引数の値を表示する
- print 自体は nil を返す

- 表示することと値を返すことを混同しないこと
- 表示には print 以外にも p や puts などがある

データの表示: p

- p 式
- ruby -e 'p 1' 1 整数
- ruby -e 'p 1+1' 2 整数な式
- ruby -e 'p "a"*3' "aaa" 文字列
- ruby -e 'p [1, 2+3, "z"]' [1,5,"z"] 配列
- ruby -e 'p true' true 真
- ruby -e 'p false' false 偽
- ruby -e 'p nil' nil nil
- ruby -e 'p /abc/' /abc/ 正規表現

正規表現

- 文字列照合の道具
- 文字列に対し以下のようなことを調べられる
 - 内部に cde という文字列が含まれているか?
 - c が3個以上並んでいて、
その次に d があるところがあるか?
 - bc という文字列か、
de という文字列か、
すくなくともどちらかは含まれているか?
 - abc の繰り返しか?
- このようなことが成立したら、
「正規表現がマッチした」という

正規表現マッチの例

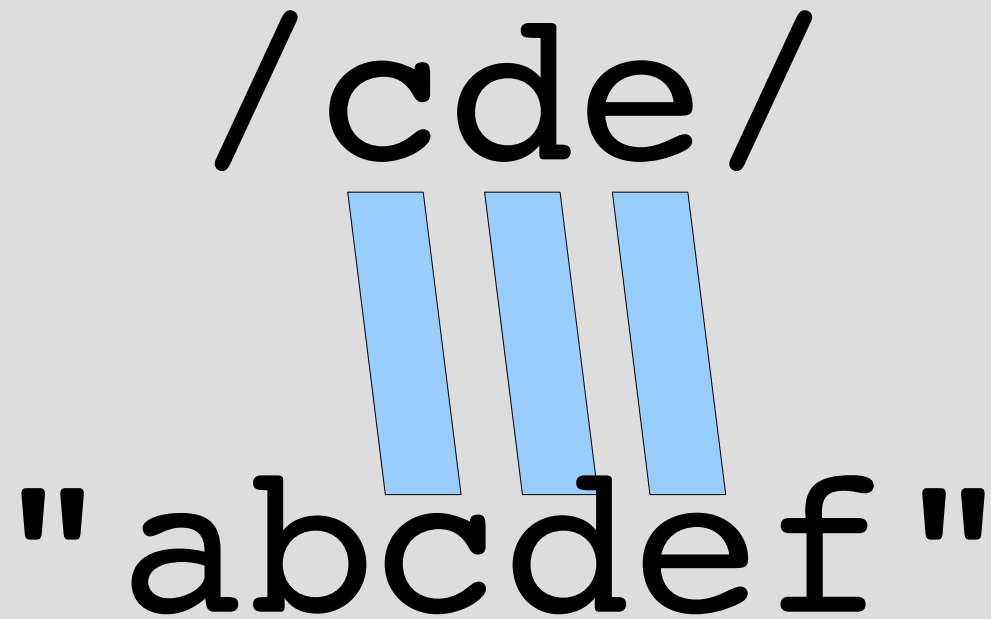
- 文字列 "abcdef" の中に cde が含まれているか?
ruby -e 'p /cde/ =~ "abcdef"'
2 "abcdef" の 2文字目からマッチ
(0-origin なので 3文字目ではない)
cde は含まれている
cde の最初の文字が 2文字目
- 文字列 "abcdef" の中に xyz が含まれているか?
ruby -e 'p /xyz/ =~ "abcdef"'
nil マッチしない
xyz は含まれていない

正規表現マッチ

- /正規表現/ =~ 文字列
- マッチしたらマッチした位置を返す
 - 文字列の先頭からマッチしたら 0 を返す (0文字目)
 - 次の文字からマッチしたら 1 を返す (1文字目)
 - 以下同様
- マッチしなかったら nil を返す

/cde/ = ~ "abcdef"

/cde/
// //
"abcdef"

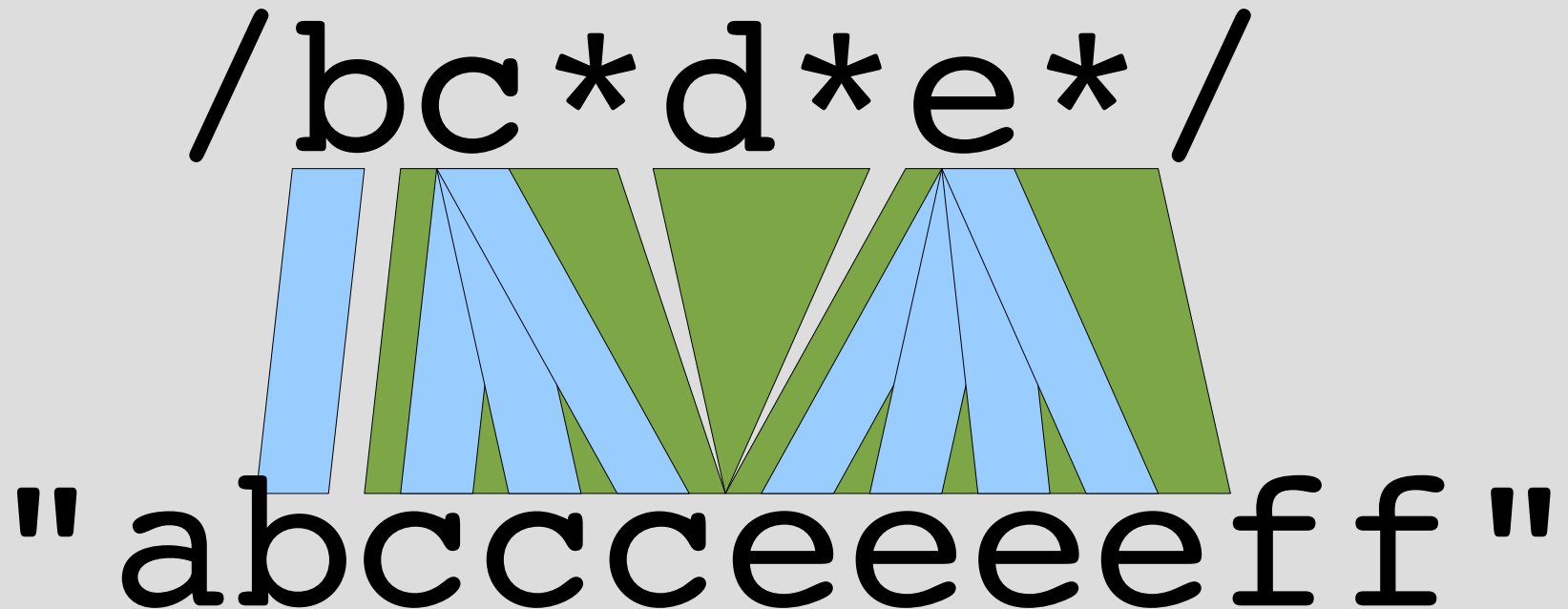


正規表現の要素

- 文字 c
- 接続 rr
- 繰り返し r^*
- 選択 $r|r$
- 文字列の先頭 $\$A$
- 文字列の最後 $\$z$
- グループ化 (r)
- 改行以外の一文字 $.$
- 他にもいろいろな要素がある

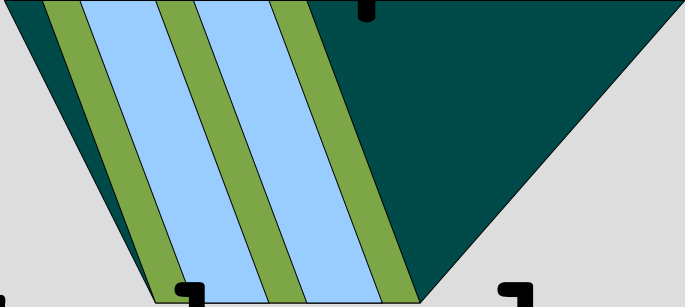
繰り返し /bc*d*e*/ =
"abcccceeeeff"

- r^* は r の 0 回以上の繰り返しにマッチする



選択 `/bc|cd/` \approx "abcde"

- `r1|r2` は `r1` と `r2` のどちらかにマッチする

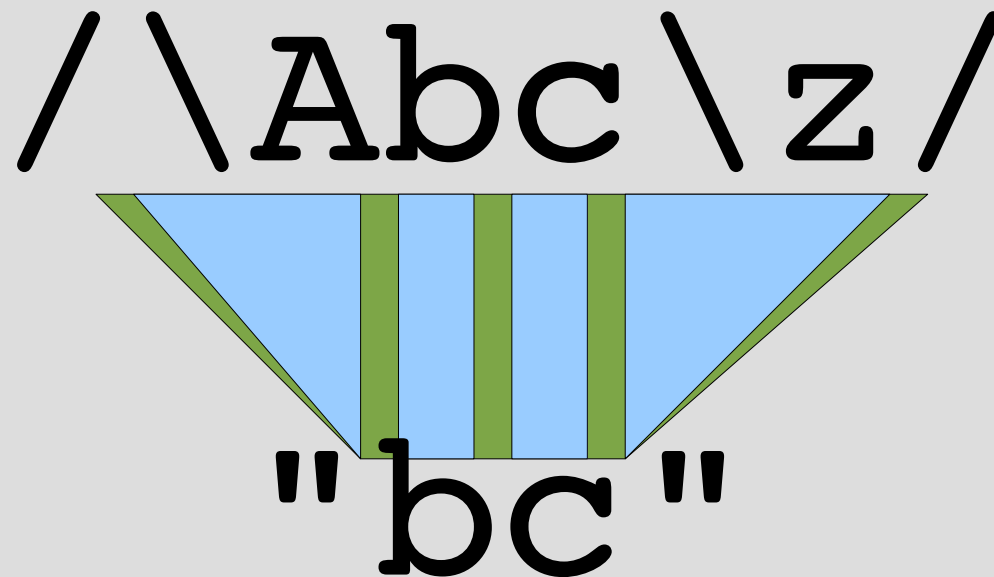
`/bc|cd/`

"abcde"

- 文字列の左から試していくので `cd` はマッチしない

文字列の先頭・最後

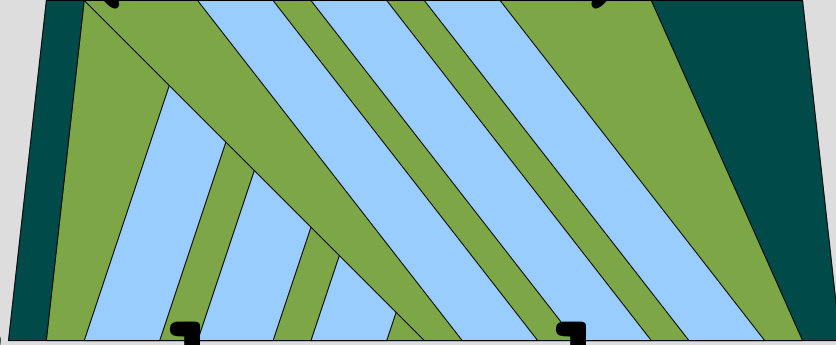
`/¥Abc¥z/ = ~ "bc"`

- ¥A は文字列の先頭にしかマッチしない
- ¥z は文字列の最後にしかマッチしない
- `/¥Abc¥z/ = ~ "bc"` はマッチする (0 を返す)
- `/¥Abc¥z/ = ~ "abcd"` はマッチしない (nil を返す)



グループ化 $/(abc)^*/ = \sim "abcabc"$

- (r) は r にマッチする

$/(abc)^*/$

"abcabc"

繰り返し /b(cd)*(ee)* / =
"abcdcdeeeeff"

- r^* の r は 1文字でなくてもよい

/b(cd)*(ee)* /
"abcdcdeeeeff"

いくらでも組合せ可能

- $\forall A((a|b)^*|(de|f^*g)^*)$
- $abc(abc)^*\forall z$
- $(\forall Aabc)|(def\forall z)$
- etc.

- 疑問:
表現できないものはあるか?
なにが表現できてなにが表現できないのだろうか?
→形式言語理論が答になる

Rubyのインストール

- この講義では安定版 Ruby 1.8.6 を想定する
- 「Rubyの歩き方」が参考になる
<http://jp.rubyist.net/magazine/?FirstStepRuby>
- Windows 上で ActiveScriptRuby を使用する場
合、インストーラを実行すれば良い
<http://arton.hp.infoseek.co.jp/indexj.html>
- Unix では、多くの場合パッケージがある
- パッケージがなくても、自分でコンパイルすることができる
- それでもできなければ質問すること

まとめ

- この授業の概要
- Ruby の使いかた
- 正規表現の使いかた
- Rubyのインストール