

rdparse

```
require './rdparse'

MONTHS = {'januari' => 1,
'februari' => 2, 'mars' => 3,
'april' => 4,
'maj' => 5, 'juni' => 6, 'juli'
=> 7, 'augusti' => 8, 'september'
=> 9,
'oktober' => 10, 'november' =>
11, 'december' => 12}

class Calendar

  def initialize
    @calendarParser =
Parser.new("calendar parser") do
  @c = Time.now
  logger.level = Logger::WARN
  token(/'.*?'/) { |m| m[1..-2]
} # quoted strings
  token(/\s+/) # whitespaces
(ignored)
  token(/\d+/) { |m| m.to_i } #
integers
  token(/\w+/) { |m| m } #
alphanumerical strings
  token(/./) { |m| m } # other
characters, individually

  start :expr do
    match(:date) { [] }
    match(:meeting)
  end

  rule :date do
    match('-', Integer, String,
Integer, '-') do |_, d, m, y, _|
      @c = Time.local(y,
MONTHS[m.downcase], d)
    end
  end
end
```

rdparse (cont)

```
end
  rule :meeting do
    match(:time, '-', :time,
String) do | start, _, stop, text
|
      [text, start, stop]
    end
  end
  rule :time do
    match(Integer, '.',
Integer) do |h, _, m|
      Time.local(@c.year,
@c.month, @c.day, h, m)
    end
  end
end

def read_meetings(filename)
  result = []
  content =
IO.readlines(filename)
  content.each do |line|
    line.strip!
    if line.length > 0 then
      r =
@c.calendarParser.parse(line)
      if r != [] then
        result << r
      end
    end
  end
  result
end
```

DSL

```
class TripleStore
  def initialize
    @db = {}
  end

  def TripleStore.load(filename)
    r = new
    r.instance_eval(File.read(filename))
  end

  def
method_missing(property, arg1, arg2)
    table = @db[property.to_s]
    if table then
      table << [arg1, arg2]
    else
      @db[property.to_s] =
[[arg1, arg2]]
    end
  end

  def find(property, arg1, arg2)
    table = @db[property]
    result = []
    table.each do |pair|
      if arg1 != '' and arg2 != ''
then
        result << pair if
(pair[0]==arg1) and
(pair[1]==arg2)
        elsif arg1 != '*' then
          result << pair if
pair[0]==arg1
        elsif arg2 != '*' then
          result << pair if
pair[1]==arg2
        end
      end
    end
  end
end
```



By Dockson

cheatography.com/dockson/

Published 9th March, 2016.

Last updated 9th March, 2016.

Page 1 of 3.

Sponsored by CrosswordCheats.com

Learn to solve cryptic crosswords!

<http://crosswordcheats.com>

DSL (cont)

```
end
  result
end
end
```

Constraint network

```
require "./constraint_networks"
class Squarer
  include PrettyPrint
  attr_accessor :a, :sq_a
  attr_reader :logger
  def initialize(a,sq_a)
    @logger=Logger.new(STDOUT)
    @a,@sq_a=[a,sq_a]
    [a,sq_a].each {|x|
x.add_constraint(self)}
  end

  def to_s
    "#{a}^2 == #{sq_a}"
  end

  def new_value(connector)
    if a==connector and
a.has_value? and (not
sq_a.has_value?) then
      logger.debug("#{self} : #
{sq_a} updated")
      sq_a.assign(a.value*a.value,s
elf)
    elsif sq_a==connector and
sq_a.has_value? and (not
a.has_value?) then
      logger.debug("#{self} : #{a}
updated")
      a.assign(Math::sqrt(sq_a.valu
e),self)
    end
  self
```

Constraint network (cont)

```
end

# A connector lost its value, so
propagate this information to all
# others
def lost_value(connector)
  ([a,sq_a]-[connector]).each {
|connector|
connector.forget_value(self) }
end

end

def circle_area_network
  radius=Connector.new('radius')
  radius_sq=Connector.new('radius_s
q')
  area=Connector.new('area')
  pi=ConstantConnector.new('pi',Ma
th::PI)
  Squarer.new(radius,radius_sq)
  Multiplier.new(pi,radius_sq,area)
  [radius,area]
end

$radius,$area=circle_area_network
def circle_area(radius)
  $radius.forget_value "user" if
$radius.has_value?
  $radius.user_assign radius
  $area.value
end
```

Teori

<p>PARSER: Parsern fungerar på så sätt att den först extraherar tokens ur teckenströmmen med hjälp av regex för att sedan arbetas på med hjälp av reglerna som är specificerade under initialize. De tokens som extraheras matchas enligt reglerna och ett uttryck byggs på så sätt upp från grunden. För varje regel som matchas så körs kodblocket bredvid det för att t.ex. spara bort koden i en variabel eller för att köra någon funktion med koden som inparameter.</p>	<p>DSL vs GPL: Ett DSL är ett skräddarsytt språk designat för att lösa specifika problem. Användning för ett GPL är väldigt brett och det används för att lösa otroligt många men orelaterade uppgifter. En anledning till att man kanske skulle vilja skriva ett DSL är för att simplificera användarinmatning, möjligtvis för användare som inte är familjära med kodning. Det kan också användas för att simplificera datainput, t.ex konfiguration för ett program.</p>
---	--



By **Dockson**

cheatography.com/dockson/

Published 9th March, 2016.

Last updated 9th March, 2016.

Page 2 of 3.

Sponsored by **CrosswordCheats.com**

Learn to solve cryptic crosswords!

<http://crosswordcheats.com>

Teori (cont)

CONTINUATION: En continuation är lite av en glorifierad GOTO. Den används för att kunna hoppa till en specifik plats någonstans i koden. Continuations sparar även 'omgivningen' som den ser ut vid just det tillfället vilket gör att man kan arbeta på variabler som de såg ut då och inte nu. En anledning till att använda continuations istället för ett antal for-loopar är hur enkelt det är att ta sig tillbaka till början continuation satsen oavsett var man är i koden för tillfället, istället för att behöva breaka en massa for-satser.

LEX vs SYNT:
Lexikalisk analys är när en ström tecken tolkas inte bara som enskilda tecken men de transformeras istället till tokens. t.ex. 'int 5' blir uppdelad till både 'int' och siffran 5. Syntaktisk analys är när dessa tokens används för att verifiera att de används enligt den specificerade grammatiken. t.ex att 'if' är följt av en jämförelse och sedan en sats.

Teori (cont)

RUBY FÖR DSL: Ruby är väldigt bra när det kommer till s.k metaprogrammering och DSL är en del av just det. En anledning till att Ruby passar för ett DSL är `method_missing`, alltså metoden där man kan dynamiskt skapa en metod utan att behöva definera den tidigare. Detta leder till en väldig frihet när det kommer till syntax. `instance_eval` är en annan väldigt användbar del av Ruby när det kommer till DSL'er. Med `instance_eval` kan man skicka in ett block kod för att köras inuti en klass. En tredje fördel är metoden `send` där namnet på en metod skickas in som en symbol och då körs denna metod i klassen i fråga.



By **Dockson**

cheatography.com/dockson/

Published 9th March, 2016.

Last updated 9th March, 2016.

Page 3 of 3.

Sponsored by **CrosswordCheats.com**

Learn to solve cryptic crosswords!

<http://crosswordcheats.com>