advanced dsl's in ruby

NEAL FORD thoughtworker / meme wrangler

ThoughtWorks

14 Wall St, Suite 2019, New York, NY 10005

nford@thoughtworks.com
www.nealford.com
www.thoughtworks.com
memeagora.blogspot.com

housekeeping

ask questions anytime

download slides from nealford.com

download samples from **`github.com/nealford`**

what i cover:

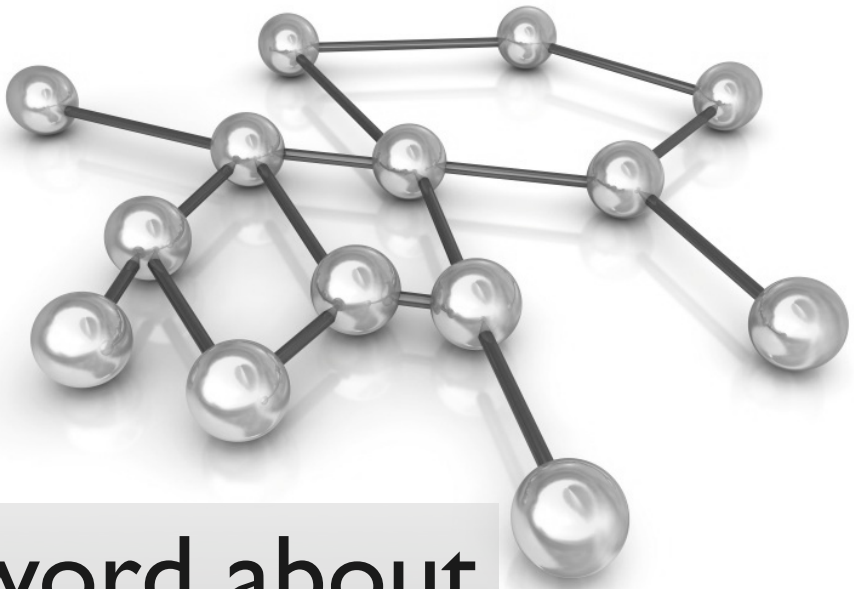
context and why it's important

building fluent interfaces

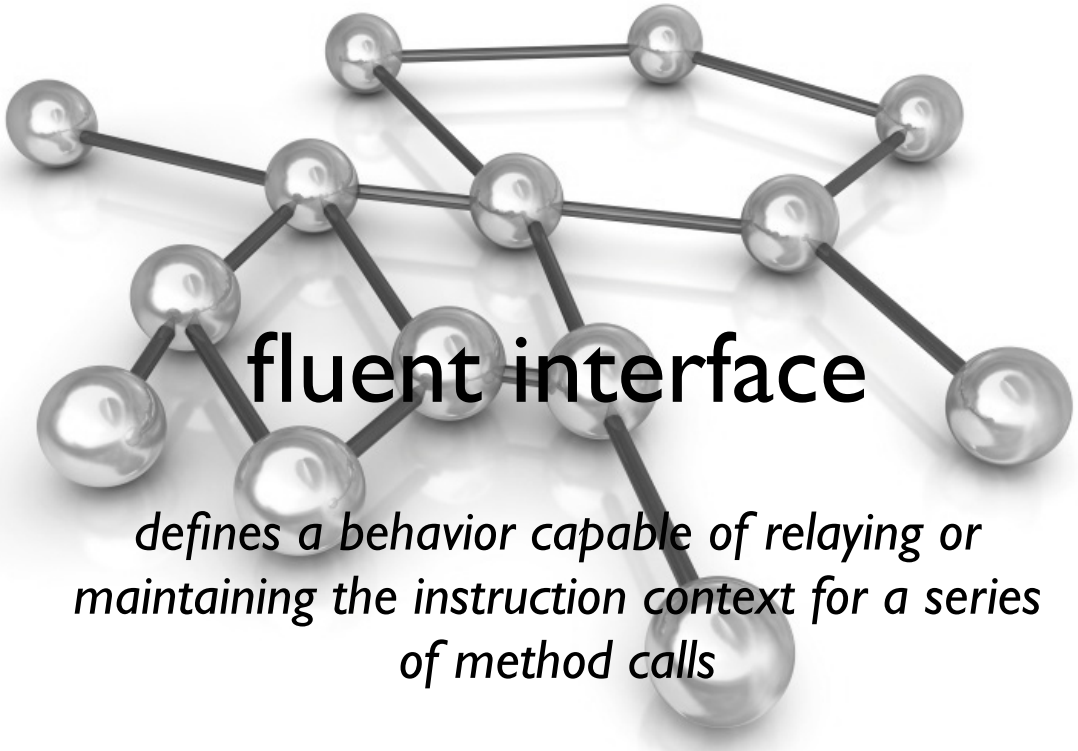
polishing, preprocessing, and parsing

business natural languages

prototype-based dsl toolkits



a word about patterns



fluent interface

defines a behavior capable of relaying or maintaining the instruction context for a series of method calls

fluent interfaces

context conveyed through:

the return value of a called method

self-referential, where the new context is equivalent to the last context

wrappers: nested or functional specification

bakery life

competition is brutal!

incentives to encourage repeat customers...

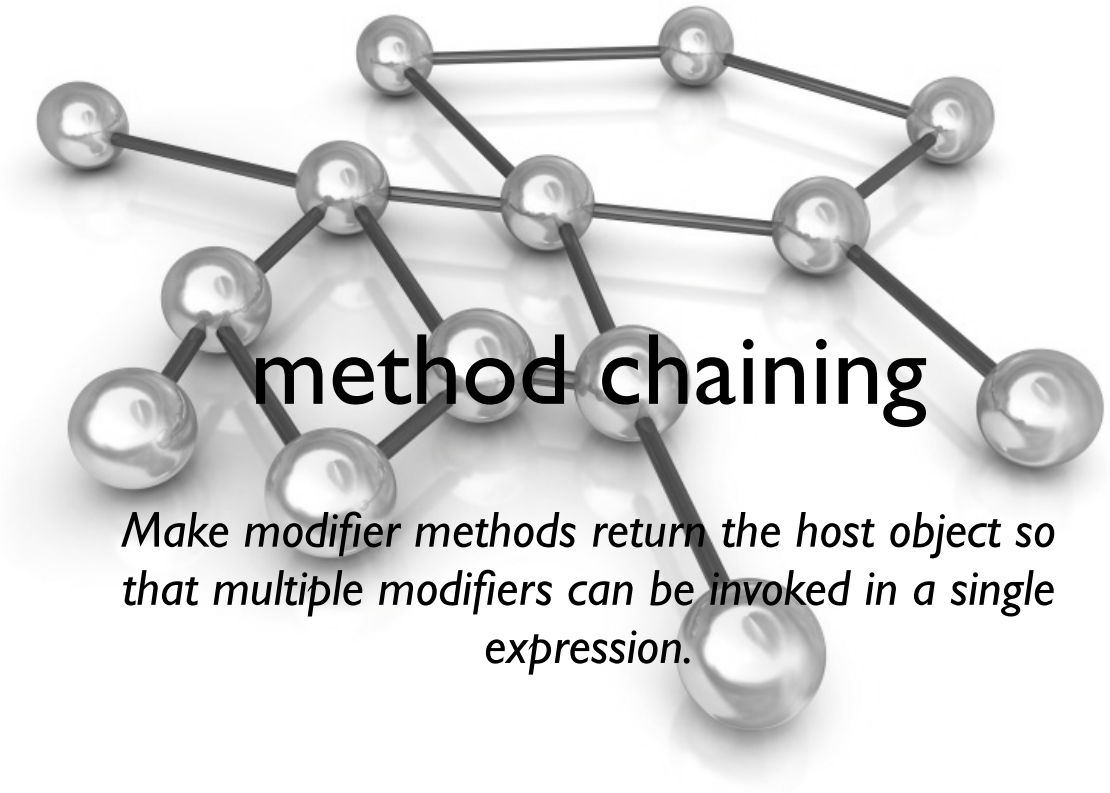
...but the other guys do the same thing

flexible business rules

easy to define & change

establishing profiles

```
comes_in_rarely = CustomerProfile.new.
    frequency(5).
    monthly_spending(20)

everyday = CustomerProfile.new.
    member.
    frequency(25).
    monthly_spending(500)
```



method chaining

Make modifier methods return the host object so that multiple modifiers can be invoked in a single expression.

discounts

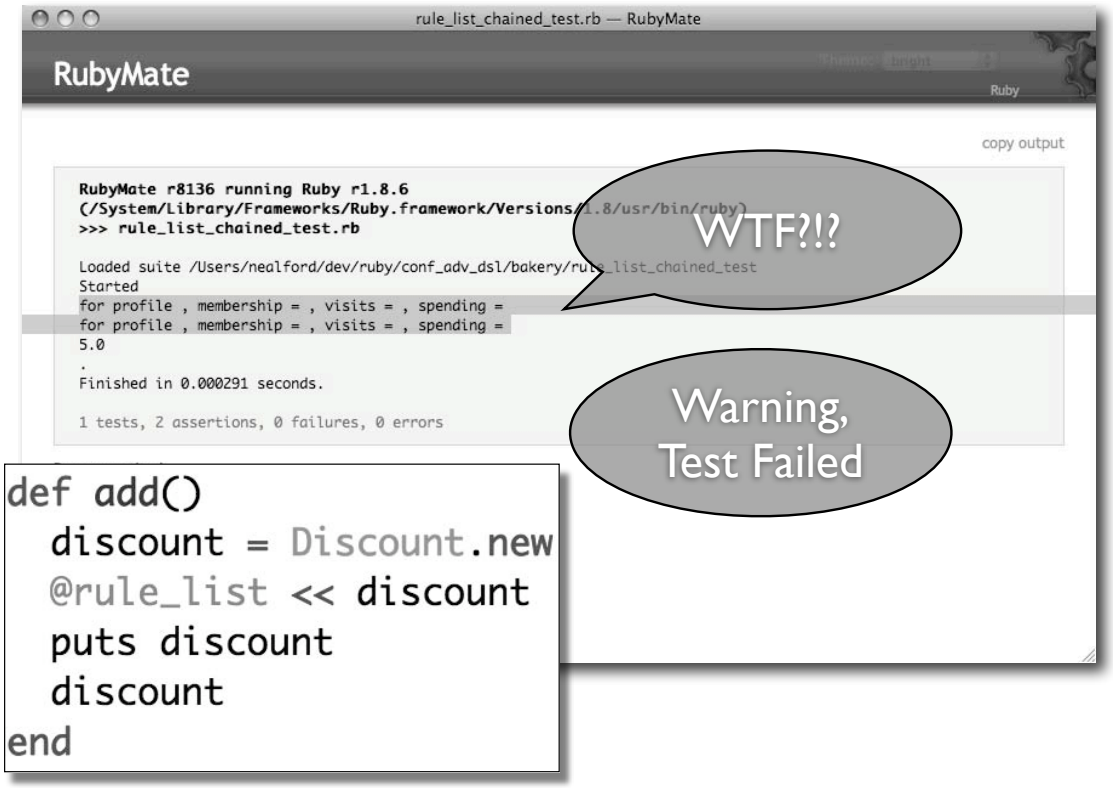
```
rules.add.
    based_on(comes_in_rarely).
    for_membership(5.0).
    for_spending(15, 5.0).
    for_number_of_visits(10, 5.0)

rules.add.
    based_on(everyday).
    for_membership(10.0).
    for_spending(100, 10.0).
    for_number_of_visits(20, 10.0)
```

```ruby
class CustomerProfile
  attr_reader :member_value, :frequency_value, :monthly_spending_value

  def initialize
    @member_value = false
  end

  def member
    @member_value = true
    self
  end

  def frequency(number_of_visits)
    @frequency_value = number_of_visits
    self
  end

  def monthly_spending(spending)
    @monthly_spending_value = spending
    self
  end
```

```ruby
class Discount
  attr_reader :discount_for_membership, :discount_for_number_of_visits,
      :discount_for_spending, :visits, :spending

  def based_on(profile)
    @profile = profile
    self
  end

  def for_membership(discount)
    @discount_for_membership = discount
    self
  end

  def for_number_of_visits(visits, discount)
    @visits = visits
    @discount_for_number_of_visits = discount
    self
  end

  def for_spending(amount, discount)
    @spending = amount
    @discount_for_spending = discount
    self
  end
```

```
class RuleListChained
   attr_reader :rule_list

   def initialize
     @rule_list = []
   end

   def add()
     discount = Discount.new
     @rule_list << discount
     puts discount
     discount
   end

   def count
     @rule_list.size
   end

   def [](index)
     @rule_list[index]
   end
end
```

```
def test_rule_list
  rules = RuleListChained.new
  comes_in_rarely = CustomerProfile.new.
      frequency(5).
      monthly_spending(20)
  everyday = CustomerProfile.new.
      member.
      frequency(25).
      monthly_spending(500)

  rules.add.
      based_on(comes_in_rarely).
      for_membership(5.0).
      for_spending(15, 5.0).
      for_number_of_visits(10, 5.0)
  rules.add.
      based_on(everyday).
      for_membership(10.0).
      for_spending(100, 10.0).
      for_number_of_visits(20, 10.0)

  assert_equal 2, rules.count
  assert_equal 5.0, rules[0].discount
end
```




```
def add()
  discount = Discount.new
  @rule_list << discount
  puts discount
  discount
end
```

why did it fail?

```
def add()
  discount = Discount.new
  @rule_list << discount
  database.put discount
  discount
end
```

the finishing problem

```
rules.add.
    based_on(comes_in_rarely).
    for_membership(5.0).
    for_spending(15, 5.0).
    for_number_of_visits(10, 5.0).
    save
```

```
class RuleList
  attr_reader :rule_list

  def initialize
    @rule_list = []
  end

  def add(discount)
    @rule_list << discount
    self
  end

  def [](index)
    @rule_list[index]
  end

  def count
    @rule_list.size
  end
end
```

```
comes_in_rarely = CustomerProfile.new
    frequency(5).
    monthly_spending(20)
everyday = CustomerProfile.new.
    member.
    frequency(25).
    monthly_spending(500)

rules.add(Discount.new.
    based_on(comes_in_rarely).
    for_membership(5.0).
    for_spending(15, 5.0).
    for_number_of_visits(10, 5.0))
rules.add(Discount.new.
    based_on(everyday).
    for_membership(10.0).
    for_spending(100, 10.0).
    for_number_of_visits(20, 10.0))
```

use method chaining for stateless object construction

use nested methods to control completion

the goal

```
recipe = Recipe.new "Spicy bread"
recipe.add 200.grams.of Flour
recipe.add 1.lb.of Nutmeg
```

open classes

```
class Numeric
  def gram
    self
  end
  alias_method :grams, :gram

  def pound
    self * 453.59237
  end
  alias_method :pounds, :pound
  alias_method :lb, :pound
  alias_method :lbs, :pound
end
```

recipe redux

```
recipe = Recipe.new "Spicy bread"
recipe.add 200.grams.of Flour
recipe.add 1.lb.of Nutmeg
```

of

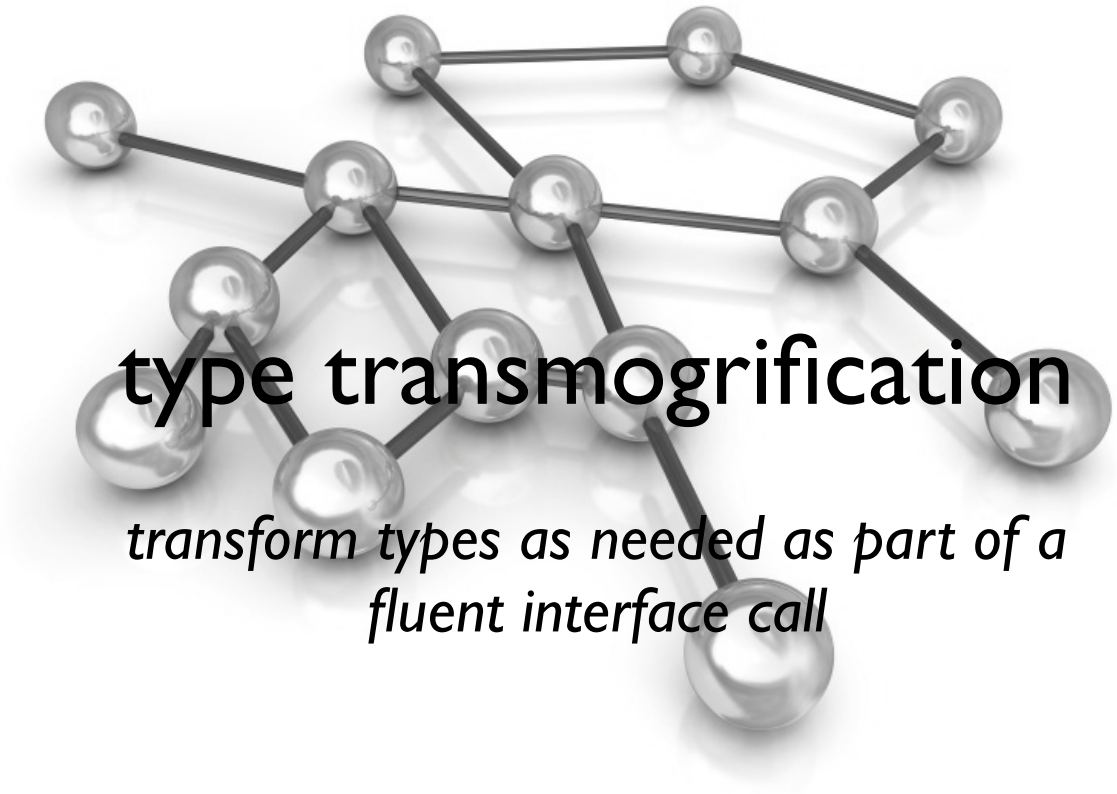
```
class Numeric
  def of ingredient
    if ingredient.kind_of? String
      ingredient = Ingredient.new(ingredient)
    end
    ingredient.quantity = self
    ingredient
  end
end
```

who returns what?

Numeric Ingredient

```
1.pound.of("Flour")
```

Integer Ingredient



type transmogrification

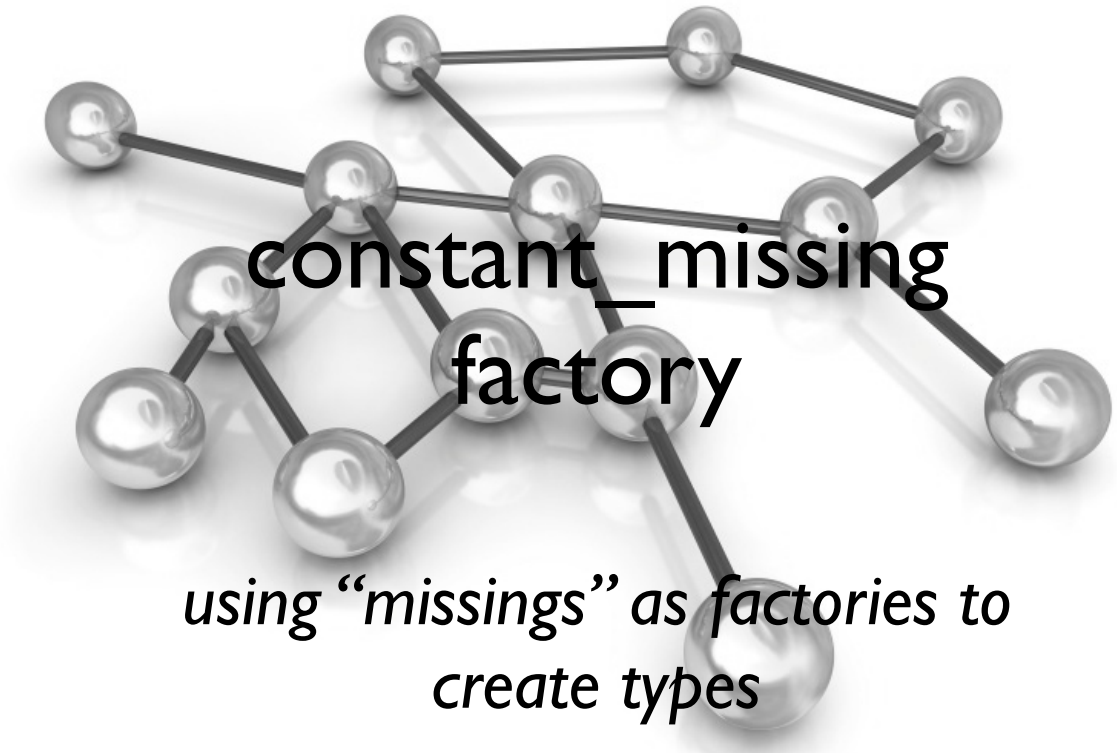
transform types as needed as part of a fluent interface call

killing noise characters

```
recipe.add 200.grams.of Flour
recipe.add 1.lb.of Nutmeg
```

const_missing

```
class Object
  def self.const_missing(sym)
    eval "Ingredient.new(sym.to_s)"
  end
end
```



constant_missing factory

using "missings" as factories to create types

ingredient factory

yikes!

```
class Object
  def self.const_missing(sym)
    Ingredient.new(sym.to_s)
  end
end
```

mix it in

```
module IngredientBuilder
  def self.append_features(target)
    def target.const_missing(name)
      Ingredient.new(name.to_s)
    end
    super
  end
end
```

safer `const` factories

```
class TestIngredients < Test::Unit::TestCase
  include IngredientBuilder

  def test_ingredient_factory
    i = Flour
    assert i.kind_of? Ingredient
    assert_equal(i.name, "Flour")
  end
```

smarter `const` factories

```
module SmartIngredientBuilder
  @@INGREDIENTS = {
    "Flour" => "Flour", "Fluor" => "Flour", "Flower" => "Flour",
    "Flur" => "Flour", "Nutmeg" => "Nutmeg", "Knutmeg" => "Nutmeg"
  }
  def self.append_features(target)
    def target.const_missing(name)
      i = @@INGREDIENTS.keys.find do |val|
        name.to_s ==  val
      end
      return Ingredient.new(@@INGREDIENTS[i]) unless i.nil?
      raise "No such ingredient"
    end
    super
  end
end
```

```
class TestSmartIngredients < Test::Unit::TestCase
  include SmartIngredientBuilder

  def test_correct_spelling
    i = Flour
    assert i.kind_of? Ingredient
    assert_equal(i.name, "Flour")
  end

  def test_misspelling
    i = Flower
    assert i.kind_of? Ingredient
    assert_equal(i.name, "Flour")
  end

  def test_missing_ingredient
    assert_raise(RuntimeError) {
      i = BakingSoda
    }
  end
end
```

shotgun approach to open classes

*don't provide universe-wide access to the whacky stuff you've implemented for your **dsl***

control your context

context

implicit context tersifies **dsl**'s

context

```
def test_verbose_syntax
  recipe = Recipe.new "Milky Gravy"
  recipe.add 1.lb.of Flour
  recipe.add 200.grams.of Milk
  recipe.add 1.gram.of Nutmeg
  assert_equal 3, recipe.ingredients.size
end
```

```
def test_consists_of
  recipe = Recipe.new "Milky Gravy"
  recipe.consists_of {
    add 1.lb.of Flour
    add 200.grams.of Milk
    add 1.gram.of Nutmeg
  }
  assert_equal 3, recipe.ingredients.size
end
```

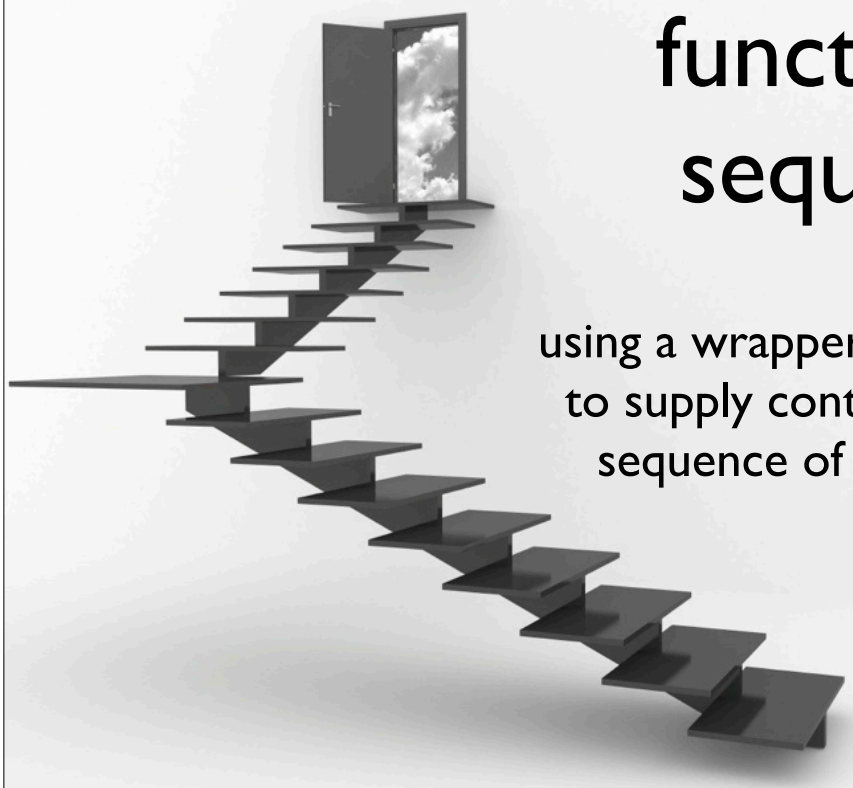
add context

```
def consists_of &block
  instance_eval &block
end
```

evaluates ruby code by switching *self* to the instance of the object calling instance_eval

context

```
def test_consists_of
  recipe = Recipe.new "Milky Gravy"
  recipe.consists_of {
    add 1.lb.of Flour
    add 200.grams.of Milk
    add 1.gram.of Nutmeg
  }
  assert_equal 3, recipe.ingredients.size
end
```



functional sequence

using a wrapper (i.e., a closure) to supply context to a linear sequence of method calls

expression builder

building a simple language for recipes allows you to build other stuff underneath

for example, a nutrition profile

recipe nutrition profile

```
def nutrition_profile
  profile = NutritionProfile.new
  ingredients.each { |i|
    foo = NutritionProfileDatabase.get_profile_for(i)
    add_to profile, NutritionProfileDatabase.get_profile_for(i)
  }
  profile
end
```

nutrition profile

```
class NutritionProfile
  attr_accessor :protein, :lipid, :sugars, :calcium, :sodium

  def initialize(protein=0, lipid=0, sugars=0, calcium=0, sodium=0)
    @protein, @lipid, @sugars =  protein, lipid, sugars
    @calcium, @sodium = calcium, sodium
  end

  def to_s()
    "\tProtein: " +   @protein.to_s       +
    "\n\tLipid: " +   @lipid.to_s         +
    "\n\tSugars: " +  @sugars.to_s        +
    "\n\tCalcium: " + @calcium.to_s       +
    "\n\tSodium: " +  @sodium.to_s
  end

end
```

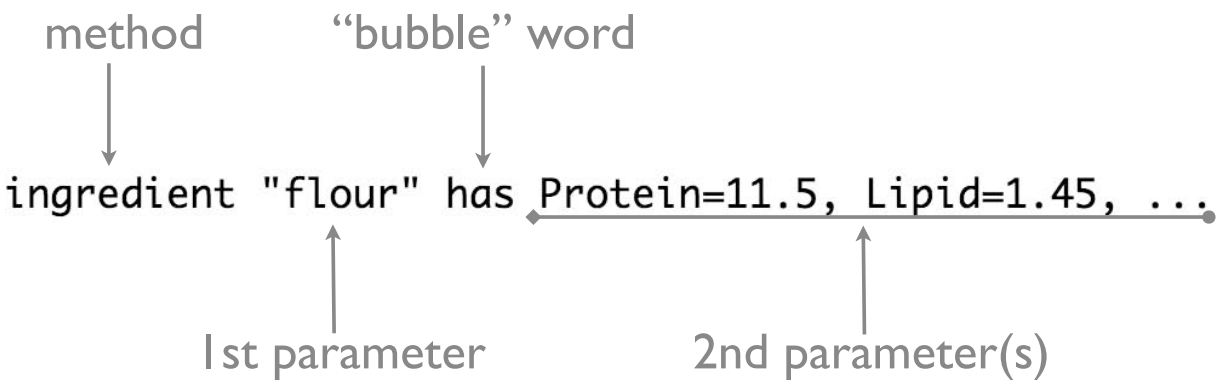
testing profile

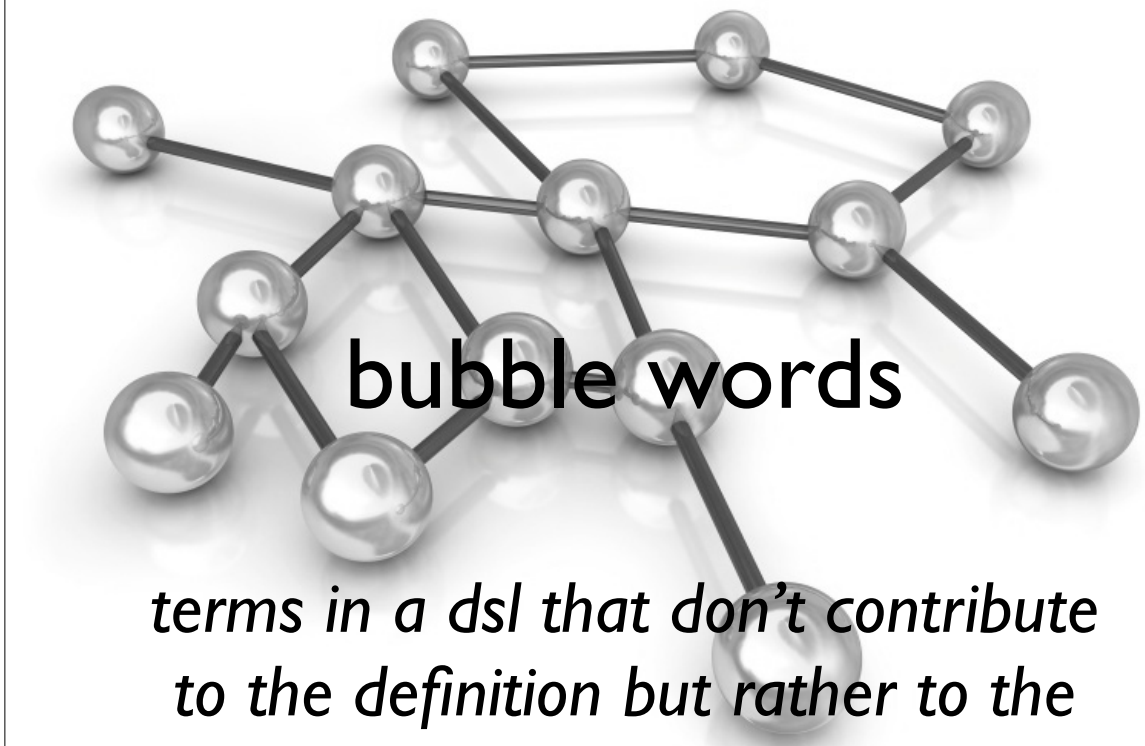
```
  def test_nutrition_profile_for_recipe
    recipe = Recipe.new
    expected = [] << 2.lbs.of(Flour) << 1.gram.of(Nutmeg)
    expected.each {|i| recipe.add i}
    protein = 11.5 + 5.84
    lipid = 1.45 + 36.31
    sugar = 1.12 + 28.49
    calcium = 20 + 184
    sodium = 2 + 16
    expected_profile = recipe.nutrition_profile
    assert_equal expected_profile.protein, protein
    assert_equal expected_profile.lipid, lipid
    assert_equal expected_profile.sugars, sugar
    assert_equal expected_profile.calcium, calcium
    assert_equal expected_profile.sodium, sodium
  end
```

profile target

```
ingredient "flour" has Protein=11.5, Lipid=1.45, Sugars=1.12, Calcium=20, Sodium=0
ingredient "nutmeg" has Protein=5.84, Lipid=36.31, Sugars=28.49, Calcium=184, Sodium=16
ingredient "milk" has Protein=3.22, Lipid=3.25, Sugars=5.26, Calcium=113, Sodium=40
```

what is this?





bubble words

terms in a dsl that don't contribute to the definition but rather to the readability

```
class NutritionProfileDefinition
  class << self
    def const_missing(sym)
      sym.to_s.downcase
    end
  end

  def ingredient(name, ingredients)
    NutritionProfile.create_from_hash name, ingredients
  end

  def process_definition(definition)
    t = polish_text(definition)
    instance_eval polish_text(definition)
  end

  def polish_text(definition_line)
    polished_text = definition_line.clone
    polished_text.gsub!(/=/, '=>')
    polished_text.sub!(/and /, '')
    polished_text.sub!(/has /, ',')
    polished_text
  end

end
```

```ruby
def test_polish_text
  test_text = "ingredient \"flour\" has Protein=11.5, Lipid=1.45, Sugars=1.12, Calcium=20, and Sodium=0"
  expected = 'ingredient "flour" ,Protein=>11.5, Lipid=>1.45, Sugars=>1.12, Calcium=>20, Sodium=>0'
  assert_equal expected, NutritionProfileDefinition.new.polish_text(test_text)
end
```

```ruby
def polish_text(definition_line)
  polished_text = definition_line.clone
  polished_text.gsub!(/=/, '=>')
  polished_text.sub!(/and /, '')
  polished_text.sub!(/has /, ',')
  polished_text
end
```

```
def process_definition(definition)
  instance_eval polish_text(definition)
end
```

```
'ingredient "flour" ,Protein=>11.5, Lipid=>1.45,
```

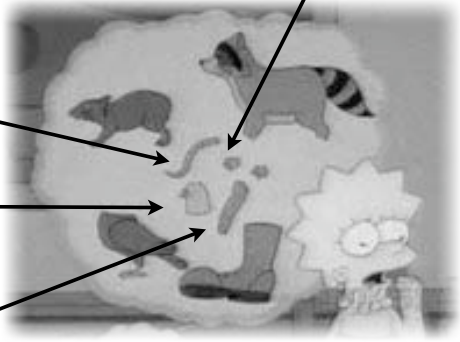
```
def ingredient(name, ingredients)
  NutritionProfile.create_from_hash name, ingredients
end
```

```
def test_create_ingredient
  actual = NutritionProfileDefinition.new.ingredient "flour",
    NutritionProfileDefinition::Protein=>11.5,
    NutritionProfileDefinition::Lipid=>1.45,
    NutritionProfileDefinition::Sugars=>1.12,
    NutritionProfileDefinition::Calcium=>20,
    NutritionProfileDefinition::Sodium=>0
  assert actual.kind_of? NutritionProfile
  assert_equal "flour", actual.name
  assert_equal 11.5, actual.protein
  assert_equal 1.45, actual.lipid
  assert_equal 1.12, actual.sugars
  assert_equal 20, actual.calcium
  assert_equal 0, actual.sodium
end
```

warning! do not try to parse text using regular expressions!

=

polish vs.
preprocess vs.
parse

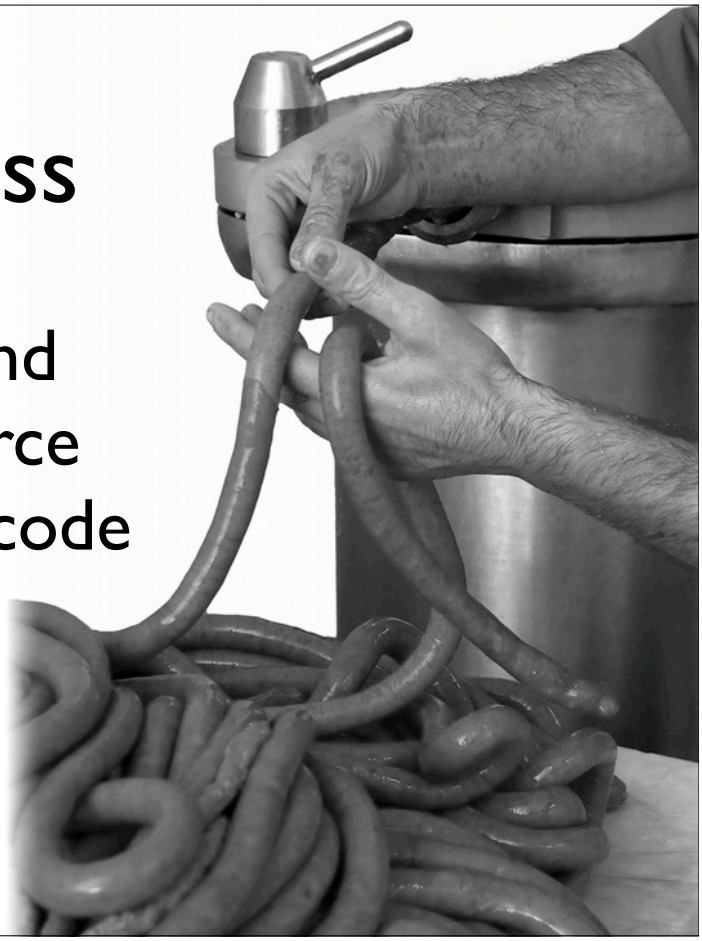


polish

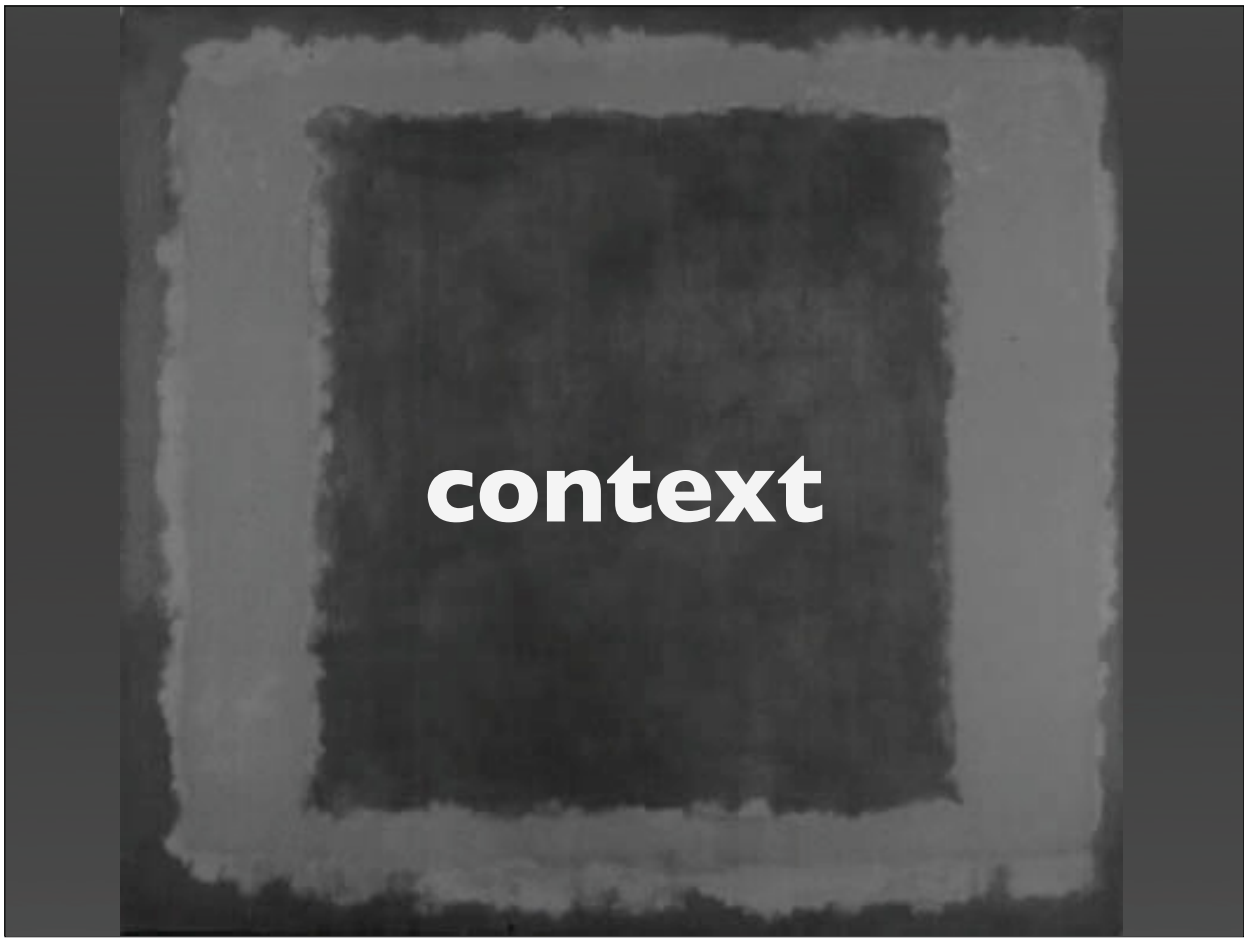
simple string substitutions to convert *nearly* ruby to *actual* ruby

pre-process

load strings and modify to coerce them into ruby code




parse
parse strings (and files) into
your own language


context

context wrapping

nested parameters

method chaining

functional sequence

- context blocks
- sticky attributes

```
require 'test/unit'
class CalculatorTest<Test::Unit::TestCase

  def test_some_complex_calculation
    assert_equal 2, Calculator.new(4).complex_calculation
  end

end
```

```ruby
class CalculatorTest<Test::Unit::TestCase

  if ENV["BUILD"] == "ACCEPTANCE"

    def test_some_complex_calculation
      assert_equal 2, Calculator.new(4).complex_calculation
    end

  end

end
```

```ruby
class CalculatorTest<Test::Unit::TestCase
  extend TestDirectives

  acceptance_only
  def test_some_complex_calculation
    assert_equal 2, Calculator.new(4).complex_calculation
  end

end
```

```ruby
class CalculatorTest<Test::Unit::TestCase
  extend TestDirectives

  acceptance_only do

    def test_some_complex_calculation
      assert_equal 2, Calculator.new(4).complex_calculation
    end

  end

end
```

```ruby
class CalculatorTest<Test::Unit::TestCase
  extend TestDirectives

  acceptance_only :test_some_complex_calculation do

    assert_equal 2, Calculator.new(4).complex_calculation

  end

end
```

```ruby
module TestDirectives

  def acceptance_only
    @acceptance_build = ENV["BUILD"] == "ACCEPTANCE"
  end

  def method_added(method_name)
    remove_method(method_name) unless @acceptance_build
    @acceptance_build = false
  end

end
```

```ruby
module TestDirectives

  def acceptance_only &block
    block.call if ENV["BUILD"] == "ACCEPTANCE"
  end

end
```

```ruby
module TestDirectives

  def acceptance_only(method_name, &method_body)
    if ENV["BUILD"] == "ACCEPTANCE"
      define_method method_name, method_body
    end
  end

end
```

```ruby
class Approval
  extend Loggable

  logged
  def decline(approver, comment)
    #implementation
  end

end
```

```
module Loggable
  def logged
    @logged = true
  end

  def method_added(method_name)
    logged_method = @logged
    @logged = false

    if logged_method
      original_method = :"unlogged_#{method_name.to_s}"
      alias_method original_method, method_name

      define_method(method_name) do |*args|
        arg_string = args.collect{ |arg| arg.inspect + " " } unless args.empty?
        log_message = "called #{method_name}"
        log_message << " with #{arg_string}" if arg_string
        Logger.log log_message
        self.send(original_method, *args)
      end
    end
  end
end
```

extant types of **dsl**s

fluent interfaces

tersifiers

implicit context

business natural languages

prototype based

business natural languages

term defined by jay fields (www.jayfields.com)

use natural language to represent business logic

bnl is a **dsl**, but not all **dsl**'s are **bnl**'s

example

```
employee John Jones

compensate $2500 for each deal closed in the past 30 days
compensate $500 for each active deal that closed more than 365 days ago
compensate 5% of gross profits if gross profits are greater than $1,000,000
compensate 3% of gross profits if gross profits are greater than $2,000,000
compensate 1% of gross profits if gross profits are greater than $3,000,000
```

process_payroll.rb

```
Dir[File.dirname(__FILE__) + "/*.bnl"].each do |bnl_file|

  vocabulary = CompensationVocabulary.new(File.basename(bnl_file, '.bnl'))

  compensation = CompensationParser.parse(File.read(bnl_file), vocabulary)

  puts "#{compensation.name} compensation: #{compensation.amount}"

end
```

vocabulary.rb

```
module Vocabulary

  def phrase(name, &block)
    define_method :"_#{name.to_s.gsub(" ","_")}", block
  end

end
```

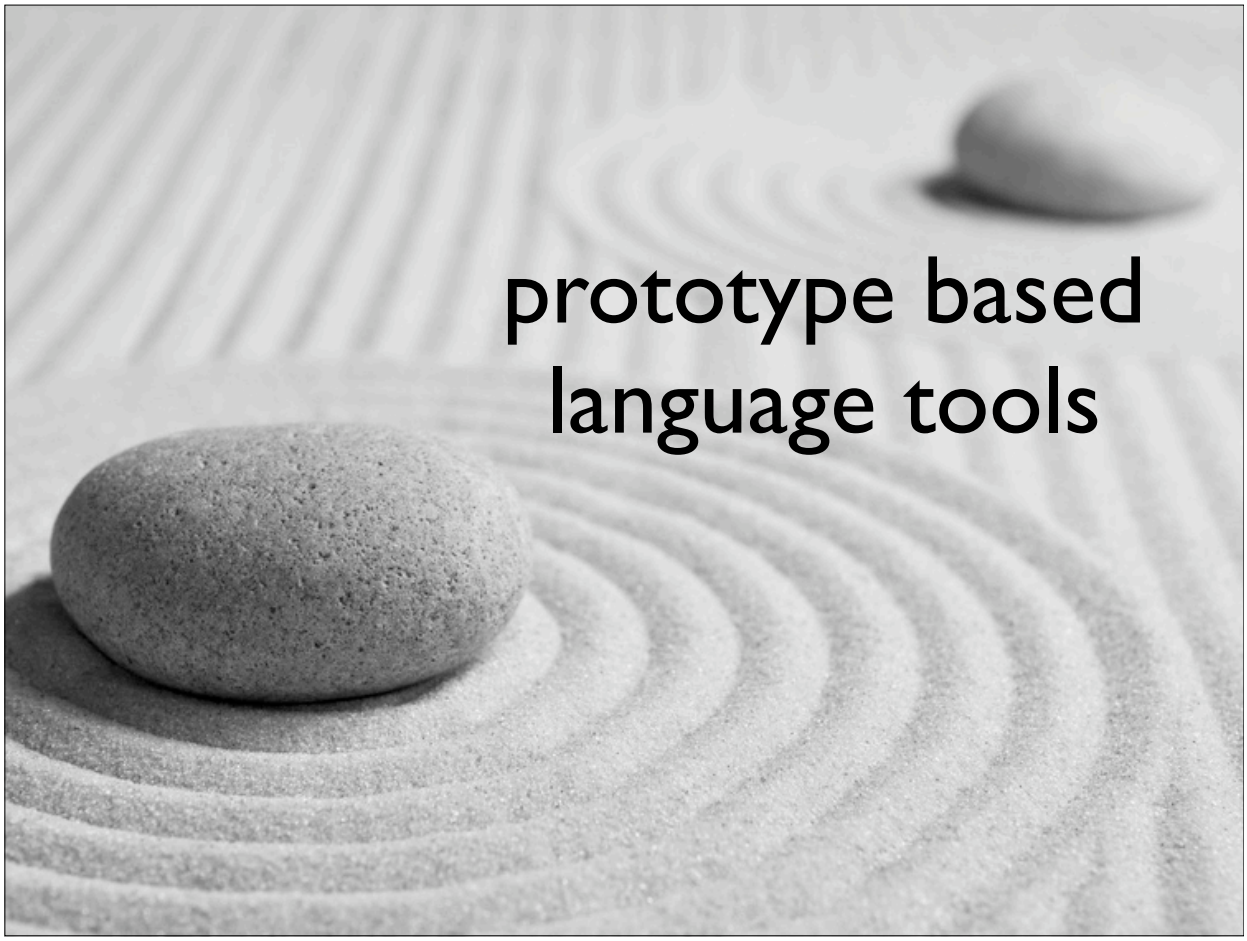
compensation_vocabulary.rb

```
class CompensationVocabulary
  extend Vocabulary

  def initialize(data_for)
    @data_for = data_for
  end

  phrase "active deal that closed more than 365 days ago!" do
    SalesInfo.send(@data_for).year_old_deals.to_s
  end

  phrase "are greater than" do
    " > "
  end

  phrase "deal closed in the past 30 days!" do
    SalesInfo.send(@data_for).deals_this_month.to_s
  end

  phrase "for each"  do
    "*"
  end
```

compensation_parser.rb

```
class CompensationParser

  class << self
    def parse(script, vocabulary)
      root = Root.new(vocabulary)
      script.split(/\n/).each { |line| root.process(preprocess(line)) }
      root
    end

    def preprocess(line)
      line.chomp!
      line.delete!('$,')
      line.gsub!(/(\d+)%/, '\1percent')
      line.gsub!(/\s/, '._')
      "_#{line.downcase}!"
    end
  end
end
```

```ruby
class Compensation

  def initialize(vocabulary)
    @phrase, @compensation_logic = '', ''
    @vocabulary = vocabulary
  end

  def method_missing(sym, *args)
    @phrase = reduce(@phrase + sym.to_s)
    if @phrase.any? && sym.to_s =~ /!$/
      raise NoMethodError.new("#{@phrase} not found")
    end
    self
  end

  def reduce(phrase)
    case
      when phrase =~ /^_\d+[(percent)!]*$/
        append(extract_number(phrase))
      when @vocabulary.respond_to?(phrase)
        append(@vocabulary.send(phrase))
      else phrase
    end
  end

  def append(piece)
    @compensation_logic += piece
    ""
  end

  def extract_number(string)
    string.gsub(/(\d+)percent$/, '0.0\1').delete('_!')
  end

  def amount
    instance_eval(@compensation_logic) || 0
  end

end
```



semr

created by matt deiters based on project work

prototype based **dsl** generator

under the radar, but open source

git://github.com/mdeiters/semr.git

semr example

```
require 'rubygems'
require 'semr'

language = Semr::Language.create do
  concept :number,    any_number, :normalize => as_fixnum
  concept :greeting,  words('hi', 'goodbye', 'hello')

  phrase 'say :greeting :number times' do |greeting, number|
    number.times { puts greeting }
  end
end

language.parse('say hello 6 times')
# hello
# hello
# hello
# hello
# hello
# hello
```

language.rb

```
module Semr
  class Language
    include Expressions
    include Normalizers

    class << self
      def create(grammer_file = nil, &block)
       language = Language.new
       language.instance_eval(&block) if block_given?
       language.instance_eval(IO.readlines(grammer_file).join("\n")) unless grammer_file.nil?
       language
      end
    end

    def concepts
     @concepts ||= {}
    end

    def phrases
     @phrases ||= []
    end
```

language.rb

```
    def concept(keyword, definition, options = {})
      concepts[keyword] = Concept.new(keyword, definition, options)
    end

    def phrase(phrase, &block)
      phrases << Phrase.new(concepts, phrase, &block)
    end

    def parse(statement)
      translation = Translation.new
      statements = statement.split('.').map{|stmt| stmt.strip } #downcase.
      statements.each do |statement|
        phrases.each do |phrase|
          if phrase.handles?(statement)
            translation.phrases_translated << phrase
            phrase.interpret(statement, translation)
            break #break loop and process next statement
          end
        end
      end
      translation
    end
  end
end
```

Xample

prototype style **dsl** processor

you give it a **dsl** (like **bnl**)...

...it generates a **dsl** processor for you

very much a work in progress (bla)

git://github.com/olabini/xample.gi

xamples

```
bonus $2,000 for each new account as of the last 12 months, payable in January
bonus $1,500 for each account with greater than 5 people in February, payable in March
bonus $1,000 for each account with greater than 10 people in March, payable in April
bonus $5,00 for each account with greater than 15 people in April, payable in May
bonus $1,000 for each account in NY, SF, or Chi every month, payable the subsequent month
bonus $1,000 for each account using C#, Java, or Ruby every month, payable the subsequent month
bonus $1,500 for each account using Erlang, Lisp, Smalltalk, or Python every month, payable the subsequent
bonus $3,000 for each account where team satisfaction is greater than 8 and the project has been running fo
bonus $2,000 for each account with a profit margin greater than 60% every month, payable the subsequent mon
bonus $100 for each consultant staffed on your accounts with a satisfaction score greater than 8 every mont
bonus $1000 for each year of employment in January, payable in May
bonus 15% of your base salary if you've been employed more than 5 years in November, payable in December
bonus 40% of your base salary if you've been employed more than 10 years in November, payable in December
bonus 5% of any profits generated from new accounts created by an employee referral each month, payable the
bonus $100 for each employee you sponsor each month, payable the subsequent month
bonus 1% of gross profit generated by your accounts for the past 12 months in January, payable in September
bonus $1000 for each account with a client satisfaction score greater than 9 and a team satisfaction score
bonus $500 for each account with no team members' satisfaction score less than 5 and the team is less than
```

summary

implicit context is everything

don't hack up the core language just to make a **dsl**

english isn't a particularly good target

tools are getting smarter

think hard about polish/preprocess/parse

?'s

please fill out the session evaluations
samples at **`github.com/nealford`**

NEAL FORD software architect / meme wrangler

ThoughtWorks

nford@thoughtworks.com
3003 Summit Boulevard, Atlanta, GA 30319
www.nealford.com
www.thoughtworks.com
memeagora.blogspot.com

resources

Text

Text

Text

Text

Text