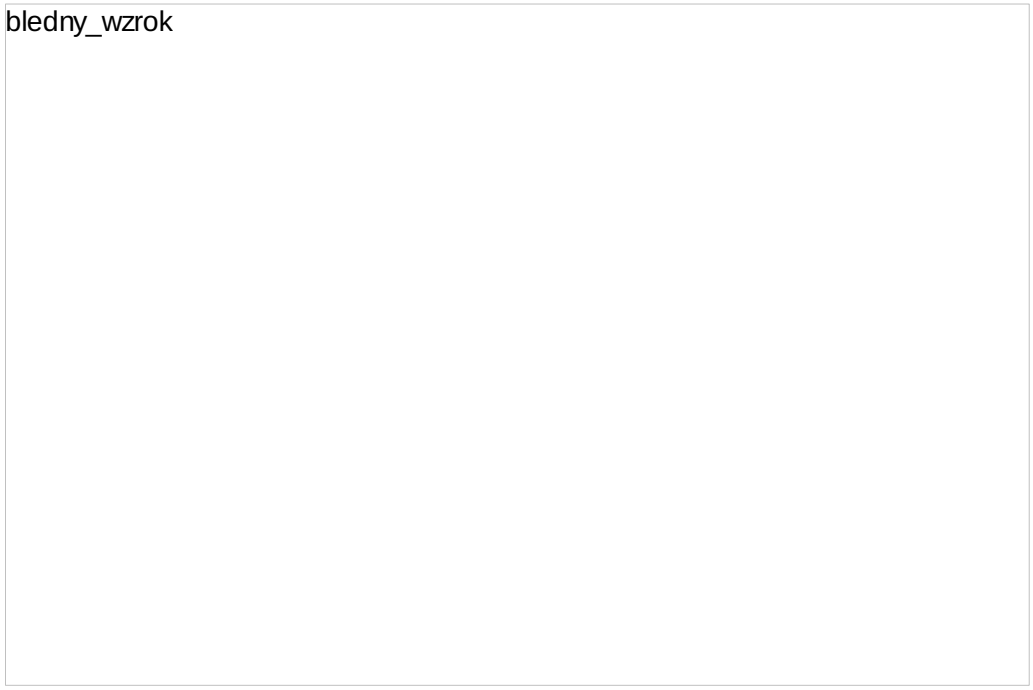
Modyfikowalny proc

Spotkało się dwóch programistów

Jeden do drugiego powiedział

Wiesz co? Chyba nie da się modyfikować Proc'a

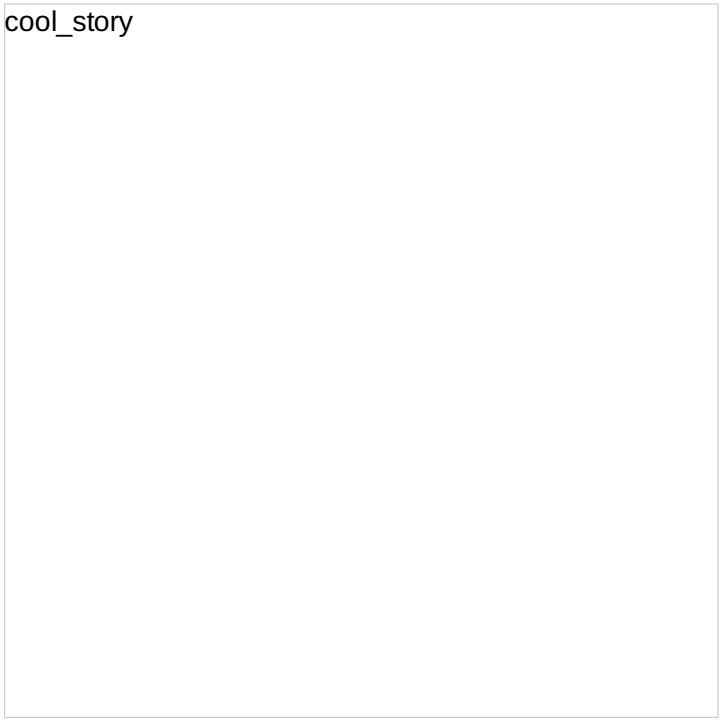

bledny_wzrok

I zaczął rozszerzać

Jaki chciał uzyskać efekt?

Wykonać coś przed lub po procu, ale w jego kontekście

```ruby
class Proc
  alias :old_initialize :initialize
  alias :old_call :call
end
```


cool_story

Pierwszy problem

Jak zachować kontekts

```ruby
class C
  def self.get_proc
    var = "proc from C"
    proc {puts var}
  end
end

var = "proc normal"
proc{ puts var }.call # => "proc normal"
C.get_proc.call # => "proc from C"
```

trzymaj_kontekst

Ratuje nas funkcja binding

```
Proc.binding # => Binding
```

```
Proc.binding # => Binding
Binding.eval
```

Jak ktoś nie lubi evali

```
class Binding
  alias :safe_eval :eval
end
```


jest_moc

Funkcja eval może jako drugi argument przyjmować binding

Drugi problem

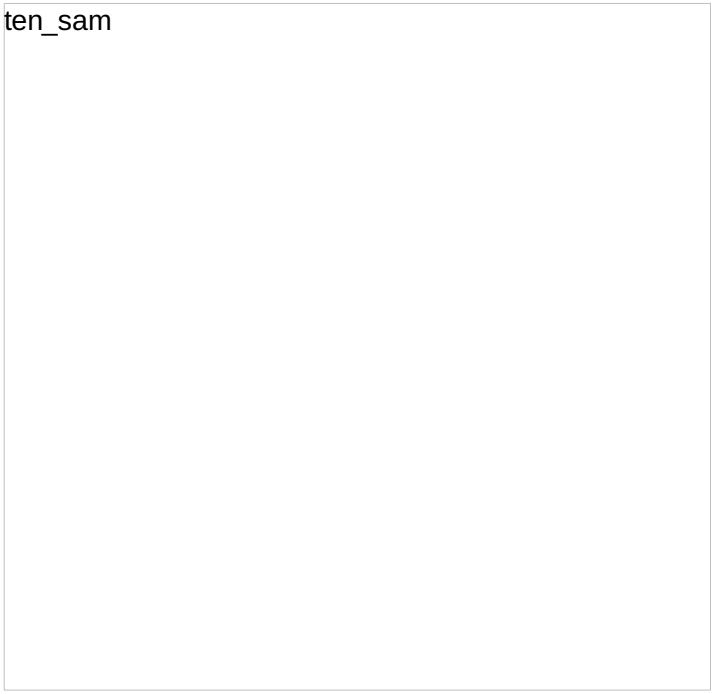
Które zmienne podmienić?

```ruby
def call(*args)
  variables =
  eval("local_variables+instance_variables",
       self.binding)

  old_call(*args)
end
```

Dodamy prostą funkcję <<

```ruby
class Proc
  def <<(other)
    if other.is_a? Proc
      @before << other
   else
     raise Exception
   end
  end
end
```

```ruby
def call(*args)
  variables =
  eval("local_variables+instance_variables",
       self.binding)
  @before.each do |proc_to_bind|
    variables.each do |var|
      # Tutaj mamy obiekt, który możemy przypisać
      var_to_bind = self.binding.eval("#{var}")
      eval("#{var} = #{var_to_bind}",
           proc_to_bind.binding) # ??????
    end
  end

  old_call(*args)
end
```


ten_sam

Jest rozwiązanie!

lambda ;-)

```ruby
def call(*args)
  variables =
  eval("local_variables+instance_variables",
       self.binding)
  @before.each do |proc_to_bind|
    variables.each do |var|
      # Tutaj mamy obiekt, który możemy przypisać
      var_to_bind = self.binding.eval("#{var}")
      eval("lambda { |v| #{var} = v }",
      proc_to_bind.binding).call(var_to_bind)
    end
  end

  old_call(*args)
end
```


fuck_yeah

```ruby
class MeineKleineKlasse
  attr_accessor :another_proc
  def initialize
    var = "another_proc"
    @another_proc = proc{puts var}
  end
end
var = "normal_proc"
normal_proc   = proc{puts var}
another_proc  = MeineKleineKlasse.new.another_proc
normal_proc << another_proc
normal_proc.call

# OUTPUT
# "normal_proc"
# "normal_proc"
```

Q & A