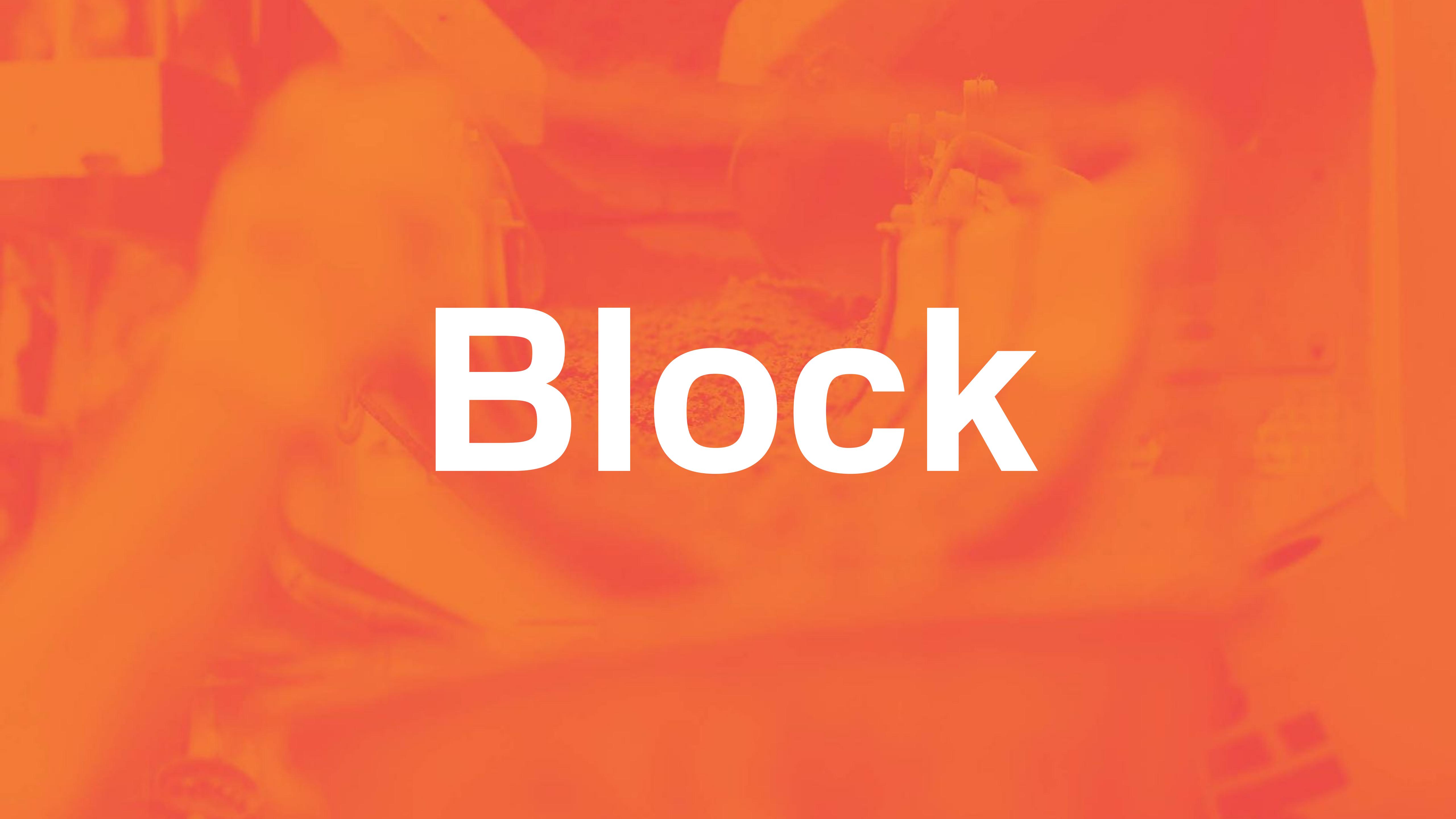


Blocks, Procs and Lambdas



Block

A Portable Chunk of Ruby Code

Passing a Block

```
greet "Hello"
```

```
greet "Hello" do
  puts "World"
end
```

```
greet "Hello" do
  puts "World"
end
```

```
greet "Hello" do
  x = rand(1..10)
  puts "big!" if x > 5
  "yolo #{x}"
end
```

```
greet "Hello" do
  puts "World"
end
```

```
greet "Hello" {  
    puts "World"  
}
```

Come to Computer Science!

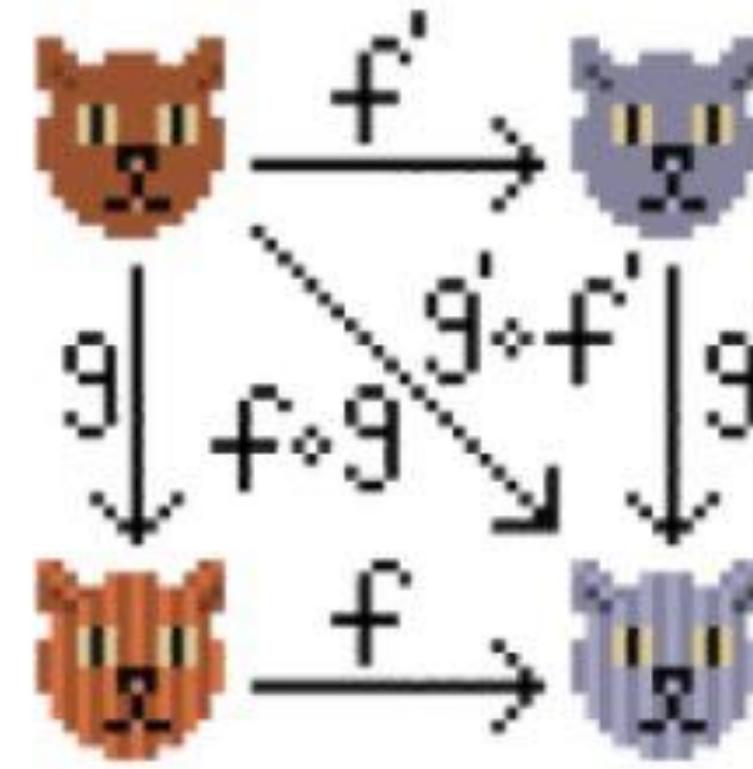
{ }

curly bois

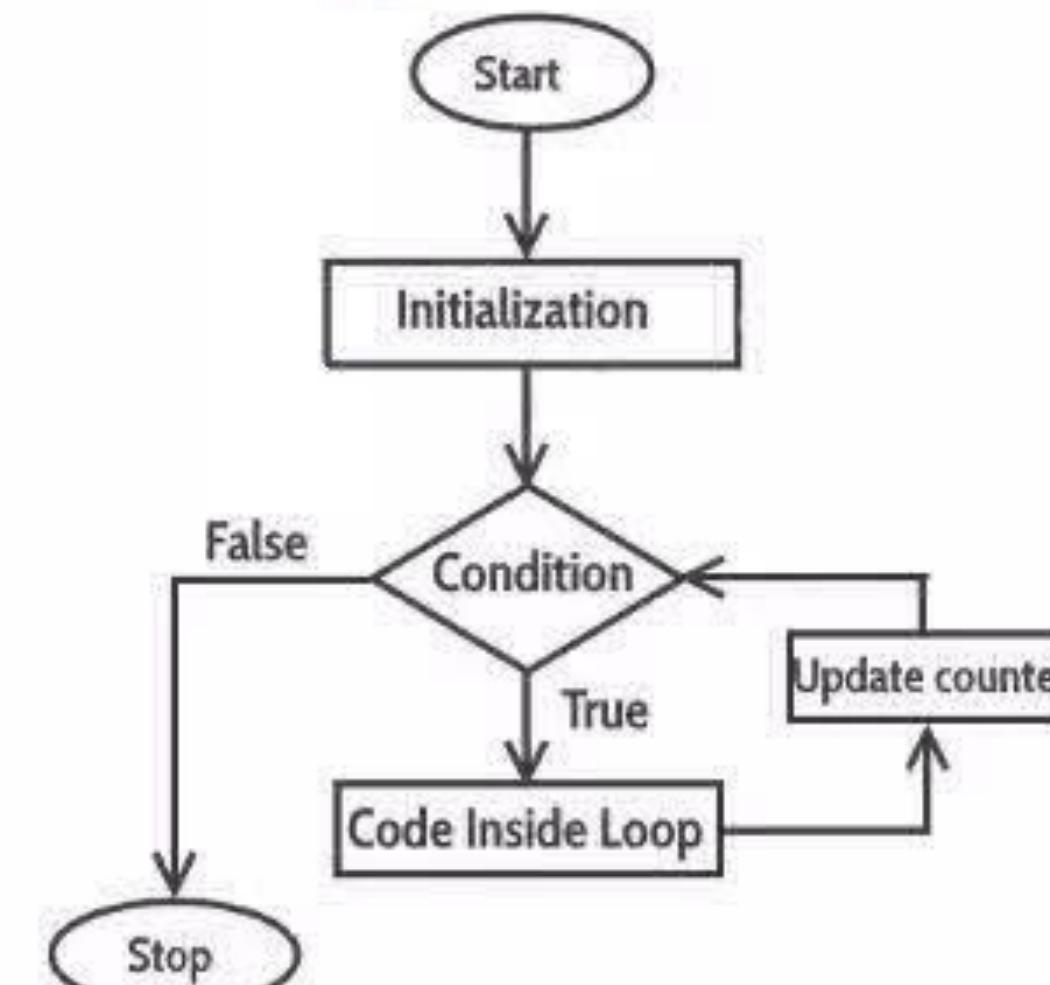
O(n!)

it large

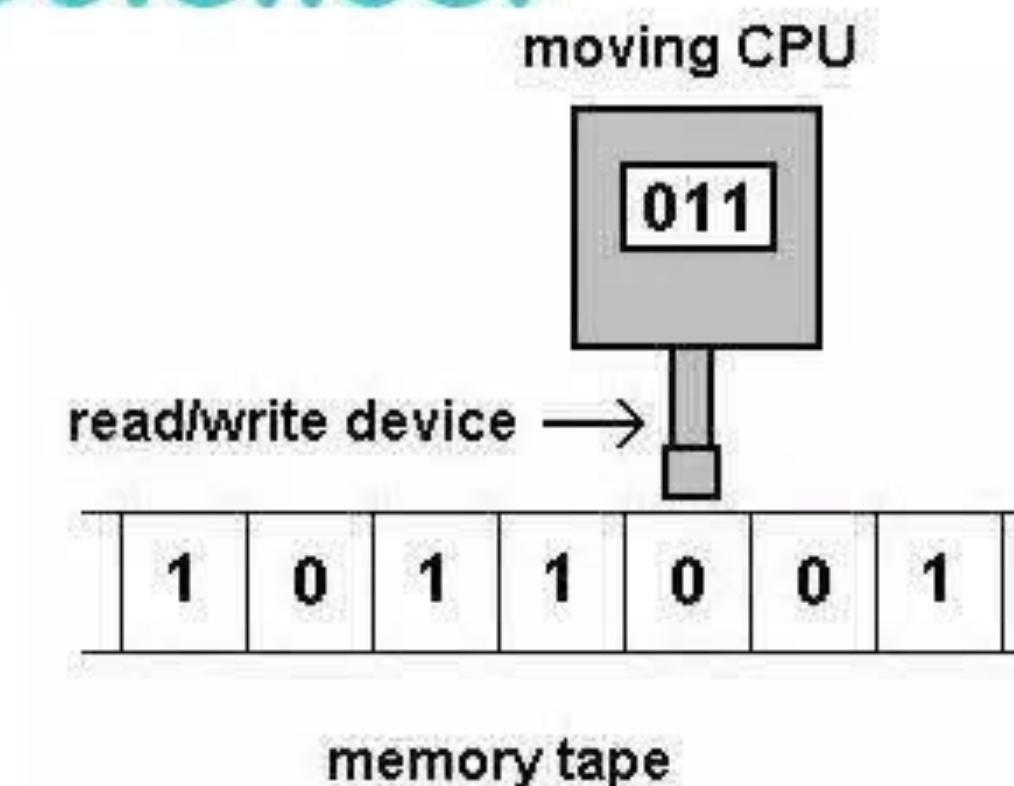
We Have...



cats



hmm fruit loops yummy



some typewriter lol idk



SNEKS

```
greet "Hello" {  
    puts "World"  
}
```

```
greet "Hello" do
  puts "World"
end
```

```
greet "Hello" do
  puts "World"
end
```

```
greet "Hello" do
  puts "World"
end
```

```
greet "Hello"
```

```
greet("Hello")
```

```
greet("Hello", do  
  puts "World"  
end)
```

```
greet("Hello") do
  puts "World"
end
```

```
greet "Hello" do
  puts "World"
end
```

Receiving a Block

```
def greet(msg)
```

```
  puts msg
```

```
end
```

```
def greet(msg)
  puts msg
  yield if block_given?
end
```

```
def greet(msg)
  puts msg
  yield if block_given?
end
```

```
def greet(msg)
  puts msg
  yield if block_given?
end
```

```
greet "Hello" do
  puts "World"
end
```

```
greet "Hello" do
  puts "World"
end
```

```
# Hello
# World
```

```
def greet(msg)
  puts msg
  yield if block_given?
  yield if block_given?
end
```

```
greet "Hello" do
  puts "World"
end
```

```
# Hello
# World
# World
```

```
greet "Hello" do
  puts "World"
end
```

```
greet "Hello" do
  "Ruby meetup!"
end
```

```
def greet(msg)
  if block_given?
    who = yield
    puts "#{msg}, #{who}!"
  else
    puts "#{msg}!"
  end
end
```

```
greet("Hello") do
  "Ruby meetup"
end
# "Hello, Ruby meetup!"
```

```
def greet(msg)
  if block_given?
    who = yield
    puts "#{msg}, #{who}!"
  else
    puts "#{msg}!"
  end
end
```

```
greet("Hello") do
  "Ruby meetup"
end
# "Hello, Ruby meetup!"
```

```
def greet(msg)
  if block_given?
    who = yield
    puts "#{msg}, #{who}!"
  else
    puts "#{msg}!"
  end
end
```

```
greet("Hello") do
  "Ruby meetup"
end
# "Hello, Ruby meetup!"
```

```
def greet(msg)
  if block_given?
    who = yield
    puts "#{msg}, #{who}!"
  else
    puts "#{msg}!"
  end
end
```

```
greet("Hello") do
  "Ruby meetup"
end
# "Hello, Ruby meetup!"
```

```
def greet(msg)
  if block_given?
    who = yield
    puts "#{msg}, #{who}!"
  else
    puts "#{msg}!"
  end
end
```

```
greet("Hello")
```

```
# "Hello!"
```

```
def greet(msg)
  if block_given?
    who = yield
    puts "#{msg}, #{who}!"
  else
    puts "#{msg}!"
  end
end
```

```
greet("Hello")
```

```
# "Hello!"
```

```
def greet(msg)
  if block_given?
    who = yield
    puts "#{msg}, #{who}!"
  else
    puts "#{msg}!"
  end
end
```

```
greet("Hello")
```

```
# "Hello!"
```

```
def greet(msg)
  if block_given?
    who = yield Location.city
    puts "#{msg}, #{who}!"
  else
    puts "#{msg}!"
  end
end
```

```
greet("Hello") do |city|
  "Ruby #{city}"
end
# "Hello, Ruby Sydney!"
```

```
def greet(msg)
  if block_given?
    who = yield Location.city
    puts "#{msg}, #{who}!"
  else
    puts "#{msg}!"
  end
end
```

```
greet("Hello") do |city|
  "Ruby #{city}"
end
# "Hello, Ruby Sydney!"
```

```
def greet(msg)
  if block_given?
    who = yield Location.city
    puts "#{msg}, #{who}!"
  else
    puts "#{msg}!"
  end
end
```

```
greet("Hello") do |city|
  "Ruby #{city}"
end
# "Hello, Ruby Sydney!"
```

```
def greet(msg)
  #
  ...
end
```

```
greet("Hello") do |city|
  "Ruby #{city}"
end
# "Hello, Ruby Sydney!"
```

```
def greet(msg)
  #
  ...
end
```

```
language = "Ruby"

greet("Hello") do |city|
  "#{language} #{city}"
end
# "Hello, Ruby Sydney!"
```

```
def greet(msg)
  #
  ...
end
```

```
language = "Ruby"
```

```
greet("Hello") do |city|
  "#{language} #{city}"
end
# "Hello, Ruby Sydney!"
```

This All Looks a
Little Bit Familiar



```
greet "Hello" do
  puts "World"
end
```

```
[1,2,3].each do |x|
  puts "Item: #{x}"
end
```

```
[1,2,3].each do |x|
  puts "Item: #{x}"
end
```

```
[1,2,3].each() do |x|
  puts "Item: #{x}"
end
```

```
[1,2,3].each do |x|
  puts "Item: #{x}"
end
```

```
[1,2,3].each do |x|
  puts "Item: #{x}"
end
```

```
[1,2,3].each do |x|
  puts "Item: #{x}"
end
```

```
[1,2,3].each do |x|
  puts "Item: #{x}"
end
```

```
greet "Hello" do
  puts "World"
end
```

```
class Array
# ...
def each
  for item in self do
    yield item if block_given?
  end
end
# ...
end
```

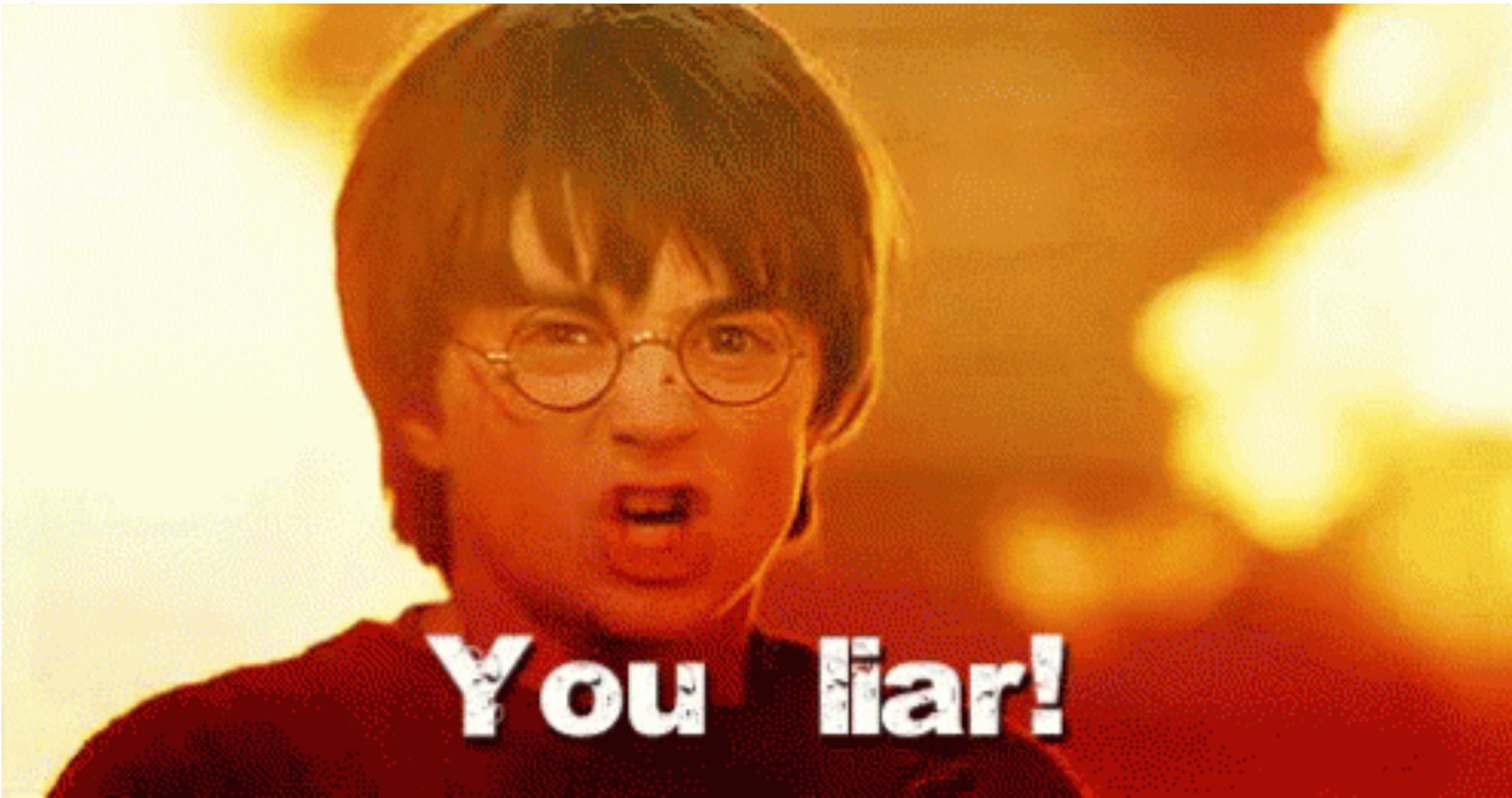
```
class Array
# ...
def each
  for item in self do
    yield item if block_given?
  end
end
# ...
end
```

```
class Array
# ...
def each
  for item in self do
    yield item if block_given?
end
end
# ...
end
```

```
class Array
# ...
def each
  for item in self do
    yield item if block_given?
  end
end
# ...
end
```

```
class Array
# ...
def each
  for item in self do
    yield item if block_given?
  end
end
# ...
end
```

class ~~Array~~ Enum



end



Lambda

```
adder = lambda do |x|  
  x + 1  
end
```

```
adder = lambda do |x|  
  x + 1  
end
```

```
adder = lambda do |x|  
  x + 1  
end
```

```
adder = lambda do |x|  
  x + 1  
end
```

```
adder = lambda do |x|  
  x + 1
```

```
end
```

```
puts adder.call(2) # 3
```

```
adder = lambda do |x|  
  x + 1  
end
```

```
puts adder.call(2) # 3
```

```
adder = lambda do |x|  
  x + 1  
end
```

```
puts adder.(2)      # 3
```

```
adder = lambda do |x|  
  x + 1  
end
```

```
puts adder[2]      # 3
```

```
adder = lambda do |x|  
  x + 1  
end
```

```
puts adder.call(2) # 3
```

```
adder = lambda do |x|  
  x + 1
```

```
end
```

```
puts adder.call(2) # 3
```

```
adder = -> (x) {  
    x + 1  
}
```

```
puts adder.call(2) # 3
```

```
adder = lambda do |x|  
  x + 1
```

```
end
```

```
puts adder.call(2) # 3
```

```
adder = lambda do |x|  
  x + 1  
end
```

```
puts adder.call(2) # 3
```

```
class Adder
  def call(x)
    x + 1
  end
end
adder = Adder.new
puts adder.call(2) # 3
```

A person wearing a vibrant red and yellow patterned dress is captured in motion, possibly dancing. They are positioned in front of a building with traditional architectural details, including a tiled roof and wooden elements. The background is slightly blurred, suggesting movement. The overall atmosphere is warm and dynamic.

Proc

```
adder = proc do |x|  
  x + 1  
end
```

```
adder = proc do |x|  
  x + 1  
end
```

```
adder = proc do |x|  
  x + 1  
end
```

Block Position



```
greet "Hello" do |city|
  "Ruby #{city}"
end
```

```
ruby = proc do |city|
  "Ruby #{city}"
end
```

```
greet "Hello", &ruby
```

```
ruby = proc do |city|
  "Ruby #{city}"
end
```

```
greet "Hello", &ruby
```

```
ruby = proc do |city|
  "Ruby #{city}"
end
```

```
greet "Hello", &ruby
```

```
ruby = proc do |city|
  "Ruby #{city}"
end
```

```
greet "Hello", &ruby
```

```
ruby = lambda do |city|
  "Ruby #{city}"
end
```

```
greet "Hello", &ruby
```

```
ruby = proc do |city|
  "Ruby #{city}"
end
```

```
greet "Hello", &ruby
```

```
def greet(msg)
  if block_given?
    who = yield Location.city
    puts "#{msg}, #{who}!"
  else
    puts "#{msg}!"
  end
end
```

```
def greet(msg)
  if block_given?
    who = yield Location.city
    puts "#{msg}, #{who}!"
  else
    puts "#{msg}!"
  end
end
```

```
def greet(msg, &block)
  if block
    who = block.call Location.city
    puts "#{msg}, #{who}!"
  else
    puts "#{msg}!"
  end
end
```



#JustProcThings

```
adder = lambda do |x|  
  x + 1  
end
```

```
puts adder.call  
# ArgumentError: wrong number of  
# arguments (given 0, expected 1)
```

```
adder = proc do |x|
  x + 1
end
```

```
puts adder.call
# NoMethodError: undefined method
# `+' for nil:NilClass
```

```
adder = proc do |x|
  x + 1
end
```

```
puts adder.call
# NoMethodError: undefined method
# `+' for nil:NilClass
```

```
adder = proc do |x|  
  x + 1  
end
```

```
puts adder.  
# NoMethodError: undefined method `+' for nil:NilClass
```



```
adder = lambda do |x|
  x + 1
end
```

```
puts adder.call(2,3)
# ArgumentError: wrong number of
# arguments (given 2, expected 1)
```

```
adder = proc do |x|  
  x + 1  
end
```

```
puts adder.call(2, 3)  
# 3
```

```
adder = proc do |x|  
  x + 1  
end
```

```
puts adder.call(2, 3)  
# 3
```

```
adder = proc do |x|  
  x + 1  
end
```

```
puts adder  
# 3
```



AND

```
def example
  puts "before"
  adder = lambda do |x|
    return x + 1
  end
  puts adder.call(2)
  puts "after"
end
```

```
def example
  puts "before"
  adder = lambda do |x|
    return x + 1
  puts "ignored"
end
puts adder.call(2)
puts "after"
end
```

```
def example
  puts "before"                      # before
  adder = lambda do |x|
    return x + 1
  puts "ignored"
end
puts adder.call(2)                   # 3
puts "after"                        # after
end
```

```
def example
  puts "before"
  adder = proc do |x|
    return x + 1
  puts "ignored"
end
puts adder.call(2)
puts "after"
end
```

```
def example
  puts "before"                                # before
  adder = proc do |x|
    return x + 1
    puts "ignored"
  end
  puts adder.call(2)                            # 3
  puts "after"
end
```

```
def example
  puts "before"                                # before
  adder = proc do |x|
    return x + 1
    puts "ignored"
  end
  puts adder.call(2)                            # 3
  puts "after"                                  # . . . ?
end
```

```
def example
  puts "before"                                # before
  adder = proc do |x|
    return x + 1
    puts "ignored"
  end
  puts adder.call(2)                            # 3
  puts "after"                                  # ...?
end
```

```
def example
  puts "before"                                # before
  adder = proc do |x|
    return x + 1
    puts "ignored"
  end
  puts adder.call(2)                            # 3
  puts "after"                                  # ...?
end
```

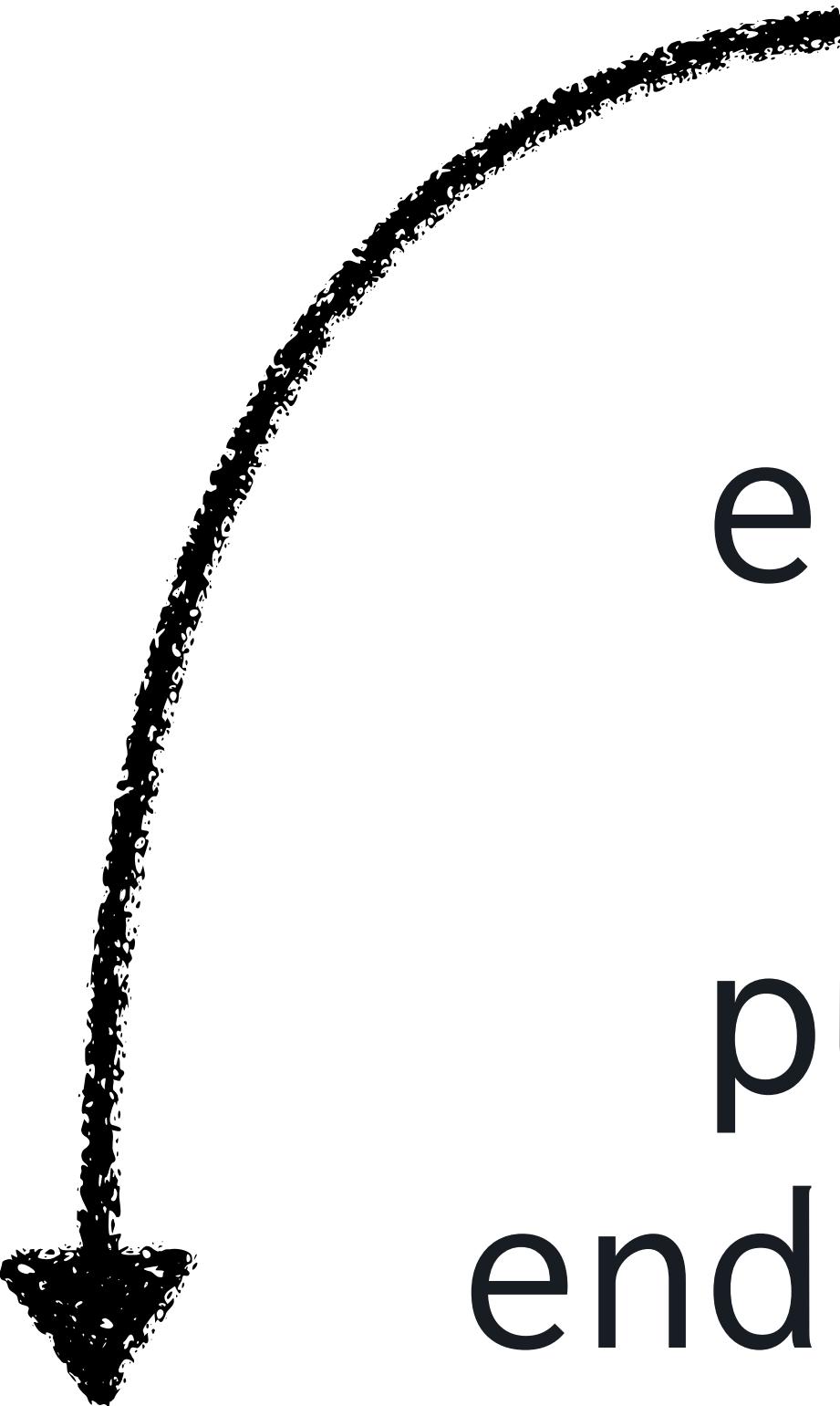


```
def example
  puts "before"                                # before
  adder = proc do |x|
    next x + 1
    puts "ignored"
  end
  puts adder.call(2)                            # 3
  puts "after"                                  # after
end
```

```
def example
  puts "before"
  [1].each do |x|
    return x + 1
  puts "ignored"
end

  puts "after"
end
```

```
def example
  puts "before"
  [1].each do |x|
    return x + 1
  puts "ignored"
end
```



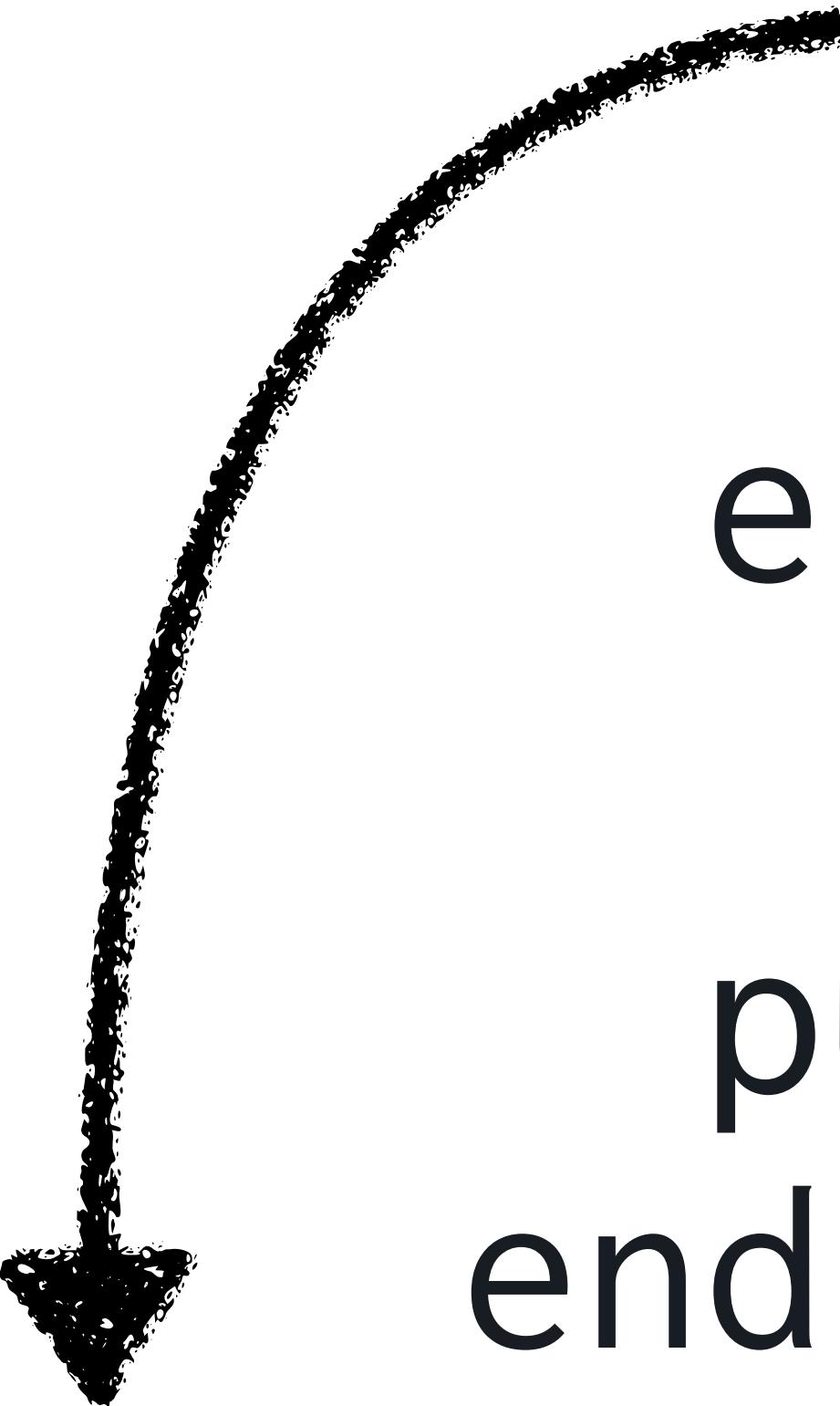
```
puts "after"
end
```

waitaminute

```
def example
  puts "before"                                # before
  adder = proc do |x|
    return x + 1
    puts "ignored"
  end
  puts adder.call(2)                            # 3
  puts "after"                                  # ...?
end
```



```
def example
  puts "before"
  [1].each do |x|
    return x + 1
  puts "ignored"
end
```



```
puts "after"
end
```

Blocks

are given to
methods as

Procs



```
> p_rock = proc { }
=> #<Proc:...@(irb):1>
```

```
> p_rock = proc { }
=> #<Proc:...@(irb):1>
```

```
> lamb_derr = lambda { }
=> #<Proc:...@(irb):2 (lambda)>
```

```
> p_rock = proc { }
=> #<Proc:...@(irb):1>
```

```
> lamb_derr = lambda { }
=> #<Proc:...@(irb):2 (lambda)>
```

```
> p_rock = proc {}  
=> #<Proc:...@(irb):1>
```

```
> lamb_derr = lambda {}  
=> #<Proc:...@(irb):2 (lambda)>
```

```
> lamb_derr.lambda?  
=> true
```

Lambdas
are
Procs





A trick

```
posts = Post.all  
posts.map do |p|  
  p.title  
end
```

```
# => [  
#   "Procs are fun",  
#   "Yay, Procs!"  
# ]
```

```
posts = Post.all  
posts.map(&:title)
```

```
# => [  
#   "Procs are fun",  
#   "Yay, Procs!"  
# ]
```

```
posts = Post.all  
posts.map(&:title)
```

```
# => [  
#   "Procs are fun",  
#   "Yay, Procs!"  
# ]
```

```
posts = Post.all  
posts.map(&:title)
```

```
# => [  
#   "Procs are fun",  
#   "Yay, Procs!"  
# ]
```



&:title

```
&:title  
=> :title.to_proc
```

```
class Symbol
# ...
def to_proc
  proc {|x| x.send(self)}
end
end
```

```
class Symbol
# ...
def to_proc
  proc {|x| x.send(self)}
end
end
```

```
class Symbol
# ...
def to_proc
  proc {|x| x.send(self)}
end
end
```

```
class Symbol
# ...
def to_proc
  proc {|x| x.send(self)}
end

end
```

```
&:title  
=> :title.to_proc
```

```
&:title  
=> :title.to_proc  
=> proc { |x| x.send(:title) }
```

```
&:title  
=> :title.to_proc  
=> proc { |x| x.send(:title) }
```

Which makes our
original call:

```
&:title  
=> :title.to_proc  
=> proc { |x| x.send(:title) }
```

```
# Which makes our  
# original call:  
posts.map(&(  
  proc{|x| x.send(:title)}  
)
```

```
posts = Post.all  
posts.map(&:title)
```

```
# => [  
#   "Procs are fun",  
#   "Yay, Procs!"  
# ]
```

A wide-angle photograph of a coastal town at sunset or sunrise. The sky is filled with warm, orange and red hues. In the center-left, a church with a tall, dark spire stands prominently. To its right, several buildings with red roofs are visible, some with laundry hanging from lines. The town is built on a hillside overlooking a body of water where a small boat can be seen. The overall atmosphere is peaceful and scenic.

Summing Up

Blocks:
Portable chunks of
Ruby code.

Procs:

**An object with a call()
method that runs a Block.**

Lambdas:
Procs (with a flag) that
has return to pretend to be
a regular method.

Ruby:
Re-open Symbol to
add `to_proc` for
#YOLO #SWAG

Blocks, Procs and Lambdas



Rob Howard
@damncabbage
<http://robhoward.id.au>