RUBY BITS

BLOCKS, PROCS, AND LAMBDAS

PRESS START

REMEMBER THIS?

```ruby
def call_this_block_twice
  yield
  yield
end
```

```ruby
call_this_block_twice { puts "tweet" }
```

```
tweet
tweet
```

What if we wanted to store this block, for execution later?

```ruby
{ puts "tweet" }
```

TWO WAYS 4 STORING BLOCKS

Proc.new

```
my_proc = Proc.new { puts "tweet" }
my_proc.call  # => tweet
```

same as

```
my_proc = Proc.new do
  puts "tweet"
end
my_proc.call  # => tweet
```

lambda

```
my_proc = lambda { puts "tweet" }
my_proc.call  # => tweet
```

Ruby 1.9

```
my_proc = -> { puts "tweet" }
my_proc.call  # => tweet
```

RUBY BITS

BLOCKS, PROCS, AND LAMBDAS

BLOCK TO LAMBDA

```ruby
class Tweet
  def post
    if authenticate?(@user, @password)
      # submit the tweet
      yield
    else
      raise 'Auth Error'
    end
  end
end
```

```ruby
class Tweet
  def post(success)
    if authenticate?(@user, @password)
      # submit the tweet
      success.call
    else
      raise 'Auth Error'
    end
  end
end
```

```ruby
tweet = Tweet.new('Ruby Bits!')
tweet.post { puts "Sent!" }
```

```ruby
tweet = Tweet.new('Ruby Bits!')
success = -> { puts "Sent!" }
tweet.post(success)
```

What about multiple lambdas?

MULTIPLE LAMBDAS

```ruby
class Tweet
  def post(success, error)
    if authenticate?(@user, @password)
      # submit the tweet
      success.call
    else
      error.call
    end
  end
end
```

```ruby
tweet = Tweet.new('Ruby Bits!')
success = -> { puts "Sent!" }
error = -> { raise 'Auth Error' }
tweet.post(success, error)
```

LAMBDA TO BLOCK

```
tweets = ["First tweet", "Second tweet"]
tweets.each do |tweet|
  puts tweet
end
```

Lets try converting to a proc

```
tweets = ["First tweet", "Second tweet"]
printer = lambda { |tweet| puts tweet }
tweets.each(printer)
```

```
ArgumentError: wrong number of
arguments (1 for 0)
```

each expects a block,
not a proc

LAMBDA TO BLOCK

```ruby
tweets = ["First tweet", "Second tweet"]
tweets.each do |tweet|
  puts tweet
end
```

Lets try converting to a proc

```ruby
tweets = ["First tweet", "Second tweet"]
printer = lambda { |tweet| puts tweet }
tweets.each(&printer)
```

'&' turns proc into block

USING THE AMPERSAND

Calling a method with & in front of a parameter

```
tweets.each(&printer)
```

turns a proc into block

Defining a method with & in front of a parameter

```
def each(&block)
```

turns a block into a proc,
so it can be assigned to parameter

Often these are used together

PASSING BLOCKS THROUGH

```
class Timeline
  attr_accessor :tweets

  def each
    tweets.each { |tweet|  yield tweet }
  end
end
```

```
timeline = Timeline.new(tweets)
timeline.each do |tweet|
  puts tweet
end
```

```
class Timeline
  attr_accessor :tweets

  def each(&block)
    tweets.each(&block)
  end
end
```

- Block into proc
- proc back into a Block

RUBY BITS

SYMBOL#TO_PROC

```ruby
tweets.map { |tweet| tweet.user }
```

same thing

```ruby
tweets.map(&:user)
```

Uses the symbol as the method name to be called.

```ruby
tweets.map(&:user.name)
```

```
undefined method `name' for
:user:Symbol (NoMethodError)
```

Cannot do this!

OPTIONAL BLOCKS

```ruby
timeline = Timeline.new
timeline.tweets = ["One", "Two"]
```

Call print without block

```ruby
timeline.print    # => One, Two
```

Call print with block

```ruby
timeline.print { |tweet|
  "tweet: #{tweet}"
}
```

```
# => tweet: First
# => tweet: Second
```

```ruby
class Timeline
  attr_accessor :tweets

  def print
    if block_given?
      tweets.each { |tweet| puts yield tweet }
    else
      puts tweets.join(", ")
    end
  end
end
```

OPTIONAL BLOCKS

```ruby
class Tweet
  def initialize
    yield self if block_given?
  end
end
```

```ruby
Tweet.new do |tweet|
  tweet.status = "Set in initialize!"
  tweet.created_at = Time.now
end
```

can optionally pass in a block that receives a tweet object

CLOSURE

```ruby
def tweet_as(user)
  lambda { |tweet| puts "#{user}: #{tweet}" }
end
```

current state of local variables is preserved when a lambda is created

```ruby
gregg_tweet = tweet_as("greggpollack")
```

resolves to

```ruby
lambda { |tweet| puts "greggpollack: #{tweet}" }
```

```ruby
gregg_tweet.call("Mind blowing!")
# => greggpollack: Mind blowing!
```

RUBY BITS