RUBY BITS

BLOCKS

PRESS START

USING BLOCKS

```
words = ['Had', 'eggs', 'for', 'breakfast.']
for index in 0..(words.length - 1)
  puts words[index]
end
```

⚠

```
words = ['Had', 'eggs', 'for', 'breakfast.']
words.each { |word| puts word }
```

✓

DECLARING BLOCKS

```
words.each { |word| puts word }
```

braces if the block is a single line

```
words.each do |word|
  backward_word = word.reverse
  puts backward_word
end
```

do/end if it's multiple lines

this is the FIRST of two conventions!

RUBY BITS

DECLARING BLOCKS

```
words.each do |word|
  puts word
end
```

do/end if the block DOES something (has a side effect)

```
backward_words = words.map { |word| word.reverse }
```

braces if you're just going to use its return value

this is the SECOND of two conventions!

YIELD

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

```ruby
call_this_block_twice { puts "twitter" }
```

```
twitter
twitter
```

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

```
tweet
tweet
```

YIELD - ARGUMENTS

```
def call_this_block
  yield "tweet"
end
```

```
call_this_block { |myarg| puts myarg }
```

········▶ tweet

```
call_this_block { |myarg| puts myarg.upcase }
```

·····▶ TWEET

RUBY BITS

YIELD - RETURN VALUE

```ruby
def puts_this_block
  puts yield
end
```

```ruby
puts_this_block { "tweet" }
```

▶ tweet

RUBY BITS

YIELD

```
def call_this_block

  block_result = yield "foo"
  puts block_result
end
```

▶ "oof"

```
call_this_block { |arg| arg.reverse }
```

USING BLOCKS

```ruby
class Timeline
  def list_tweets
    @user.friends.each do |friend|
      friend.tweets.each { |tweet| puts tweet }
    end
  end
  def store_tweets
    @user.friends.each do |friend|
      friend.tweets.each { |tweet| tweet.cache }
    end
  end
end
```

same iteration, different logic

YOUR OWN "EACH"

```ruby
class Timeline
  def each
    @user.friends.each do |friend|
      friend.tweets.each { |tweet| yield tweet }
    end
  end
end


timeline = Timeline.new(user)
timeline.each { |tweet| puts tweet }
timeline.each { |tweet| tweet.cache }
```

re-use iteration

vary logic

RUBY BITS

ENUMERABLE

```
class Timeline
  def each
    ...
  end
  include Enumerable
end
```

you implemented "each", now mix in Enumerable

```
timeline.sort_by  { |tweet| tweet.created_at }
timeline.map      { |tweet| tweet.status }
timeline.find_all { |tweet| tweet.status =~ /\@codeschool/ }
```

you instantly get all these methods, and more!

"EXECUTE AROUND"

```ruby
def update_status(user, tweet)
  begin
    sign_in(user)
    post(tweet)
  rescue ConnectionError => e
    logger.error(e)
  ensure
    sign_out(user)
  end
end
```

```ruby
def get_list(user, list_name)
  begin
    sign_in(user)
    retrieve_list(list_name)
  rescue ConnectionError => e
    logger.error(e)
  ensure
    sign_out(user)
  end
end
```

everything but the core
logic is duplicated!

"EXECUTE AROUND"

```ruby
def while_signed_in_as(user)
  begin
    sign_in(user)
    yield
  rescue ConnectionError => e
    logger.error(e)
  ensure
    sign_out(user)
  end
end
```

```ruby
while_signed_in_as(user) do
  post(tweet)
end
```

```ruby
tweets = while_signed_in_as(user) do
  retrieve_list(list_name)
end
```

now you can just call the single method with a block!

RUBY BITS

"EXECUTE AROUND"

```ruby
def while_signed_in_as(user)
  sign_in(user)
  yield
  rescue ConnectionError => e
    logger.error(e)
  ensure
    sign_out(user)
end
```

COMBO X2!

no need for begin/end
within a method!

RUBY
BITS
BLOCKS
PRESS START