
RUBY
BITS
UNDERSTANDING & USING SELF
PRESS START

SELF

```
puts "Outside the class: #{self}"

class Tweet
  puts "Inside the class: #{self}"
end
```

```
Outside the class: main
Inside the class: Tweet
```

Ruby's special top level object

the Tweet class object

SELF

```ruby
class Tweet
  def self.find(keyword)
    puts "Inside a class method: #{self}"
  end
end

Tweet.find("rubybits")
```

```
Inside a class method: Tweet
```

the Tweet class object

RUBY BITS

CLASS METHODS

```ruby
class Tweet
  def self.find(keyword)
    # do stuff here
  end
end
```

```ruby
def Tweet.find(keyword)
  # do stuff here
end
```

this is equivalent, but not often used

RUBY BITS

SELF

```ruby
class Tweet
  def initialize(status)
    puts "Inside a method: #{self}"
    @status = status
  end
end

Tweet.new("What is self, anyway?")
```

the Tweet instance

```
Inside a method: #<Tweet:0x007f8c222de488>
```

RUBY BITS

FINDING METHODS

When there's no explicit receiver, look in self

```
class Tweet
  attr_accessor :status

  def initialize(status)
    @status = status
    set_up_some_things
  end

  def set_up_some_things
    # do something here
  end
end
```

called on the Tweet class object

instance variable added to the Tweet instance

called on the Tweet instance

RUBY BITS

CLASS_EVAL

Sets self to the given class and executes a block

```ruby
class Tweet
  attr_accessor :status, :created_at

  def initialize(status)
    @status = status
    @created_at = Time.now
  end
end
```

```ruby
Tweet.class_eval do
  attr_accessor :user
end
```

inside the block, self is the Tweet class

```ruby
tweet = Tweet.new("Learning class_eval with Ruby Bits")
tweet.user = "codeschool"
```

RUBY BITS

CREATING A METHOD LOGGER

```
class Tweet
  def say_hi
    puts "Hi"
  end
end

logger = MethodLogger.new
logger.log_method(Tweet, :say_hi)
```

tell the MethodLogger to log calls to say_hi

RUBY BITS

CREATING A METHOD LOGGER

How should MethodLogger work?

log_method takes a class and method name
use class_eval to execute code in the class
    use alias_method to save the original method
    use define_method to redefine the method
        log the method call
        use send to call the original method

RUBY BITS

CREATING A METHOD LOGGER

```ruby
class MethodLogger
  def log_method(klass, method_name)
    klass.class_eval do
      alias_method "#{method_name}_original", method_name
      define_method method_name do
        puts "#{Time.now}: Called #{method_name}"
        send "#{method_name}_original"
      end
    end
  end
end
```

save the original method

log the method call

call the original method

CREATING A METHOD LOGGER

```
class Tweet
  def say_hi
    puts "Hi"
  end
end

logger = MethodLogger.new
logger.log_method(Tweet, :say_hi)

Tweet.new.say_hi
```

```
2012-09-01 12:52:03 -400: Called say_hi
Hi
```


RUBY BITS

CREATING A METHOD LOGGER

Bonus: Log methods with params and blocks

```
class MethodLogger
  def log_method(klass, method_name)
    klass.class_eval do
      alias_method "#{method_name}_original", method_name
      define_method method_name do |*args, &block|
        puts "#{Time.now}: Called #{method_name}"
        send "#{method_name}_original", *args, &block
      end
    end
  end
end
```

capture args and block

pass them to the original method

RUBY BITS

INSTANCE_EVAL

Sets self to the given instance and executes a block

```ruby
class Tweet
  attr_accessor :user, :status
end
```

```ruby
tweet = Tweet.new
tweet.instance_eval do
  self.status = "Changing the tweet's status"
end
```

....inside the block,
self is the Tweet instance

RUBY BITS

INSTANCE_EVAL

```ruby
class Tweet
  attr_accessor :user, :status

  def initialize
    yield self if block_given?
  end
end

Tweet.new do |tweet|
  tweet.status = "I was set in the initialize block!"
  tweet.user = "Gregg"
end
```

... our block needs the tweet instance

can we clean this up?

RUBY BITS

INSTANCE_EVAL

```ruby
class Tweet
  attr_accessor :user, :status

  def initialize(&block)
    instance_eval(&block) if block_given?
  end
end

Tweet.new do
  self.status = "I was set in the initialize block!"
  self.user = "Gregg"
end
```

capture the block

pass it to instance_eval

RUBY BITS