RUBY BITS

HANDLING MISSING METHODS

PRESS START

METHOD_MISSING

Called when Ruby can't find a method

```ruby
class Tweet
  def method_missing(method_name, *args)
    puts "You tried to call #{method_name} with these arguments: #{args}"
  end
end

Tweet.new.submit(1, "Here's a tweet.")
```

```
You tried to call submit with arguments: [1, "Here's a tweet."]
```

RUBY BiTS

METHOD_MISSING

```
class Tweet
  def method_missing(method_name, *args)
    logger.warn "You tried to call #{method_name} with these arguments: #{args}"
    super
  end
end

Tweet.new.submit(1, "Here's a tweet.")
```

write to our log

Ruby's default method_missing handling raises a NoMethodError

DELEGATING METHODS

```ruby
class Tweet
  def initialize(user)
    @user = user
  end

  def username
    @user.username
  end

  def avatar
    @user.avatar
  end
end
```

starting to see duplication,
and what if we need to add more of these?

DELEGATING METHODS

```
class Tweet
  def initialize(user)
    @user = user
  end

  def method_missing(method_name, *args)
    @user.send(method_name, *args)
  end
end
```

send all unknown method calls to the user

DELEGATING METHODS

```ruby
class Tweet
  DELEGATED_METHODS = [:username, :avatar]

  def initialize(user)
    @user = user
  end

  def method_missing(method_name, *args)
    if DELEGATED_METHODS.include?(method_name)
      @user.send(method_name, *args)
    else
      super
    end
  end
end
```

delegate only certain methods

default handling for all other methods

SIMPLE DELEGATOR

```
require 'delegate'

class Tweet < SimpleDelegator
  def initialize(user)
    super(user)
  end
end
```

automatically delegates all unknown methods to user

RUBY BITS

How can we add this functionality with method_missing?

```ruby
tweet = Tweet.new("Sponsored by")
tweet.hash_ruby
tweet.hash_metaprogramming
puts tweet
```

```
Sponsored by #ruby #metaprogramming
```

We can call any method with hash_*

DYNAMIC METHODS

```ruby
class Tweet
  def initialize(text)
    @text = text
  end

  def to_s
    @text
  end

  def method_missing(method_name, *args)
    match = method_name.to_s.match(/^hash_(\w+)/)
    if match
      @text << " #" + match[1]
    else
      super
    end
  end
end
```

```ruby
tweet = Tweet.new("Sponsored by")
tweet.hash_ruby
tweet.hash_metaprogramming
puts tweet
```

RESPOND_TO?

Tells us if an object responds to a given method

```
tweet = Tweet.new
tweet.respond_to?(:to_s)         # => true
tweet.hash_ruby
tweet.respond_to?(:hash_ruby) # => false
```

works because we defined method_missing

lies!!!

RESPOND_TO?

```ruby
class Tweet
  ...
  def respond_to?(method_name)
    method_name =~ /^hash_\w+/ || super
  end
end
```

respond to methods starting with hash_

default handling for everything else

```ruby
tweet = Tweet.new
tweet.respond_to?(:hash_ruby)  # => true
```

✓

```ruby
tweet.method(:hash_ruby)
```

NameError: undefined method

RUBY BITS

RESPOND_TO_MISSING?

Ruby 1.9.3

```
class Tweet
  ...
  def respond_to_missing?(method_name)
    method_name =~ /^hash_\w+/ || super
  end
end
```

```
tweet = Tweet.new
tweet.method(:hash_ruby)
```

returns a Method object as expected

DEFINE_METHOD REVISITED

```ruby
def method_missing(method_name, *args)
  match = method_name.to_s.match(/^hash_(\w+)/)
  if match
    @text << " #" + match[1]
  else
    super
  end
end
```

```ruby
tweet.hash_codeschool
tweet.hash_codeschool
```

calls method_missing both times

RUBY BITS

DEFINE_METHOD REVISITED

```
def method_missing(method_name, *args)
  match = method_name.to_s.match(/^hash_(\w+)/)
  if match
    self.class.class_eval do
      define_method(method_name) do
        @text << " #" + match[1]
      end
    end
    send(method_name)
  else
    super
  end
end
```

execute in context of the class

define a new method

then call it

```
def hash_codeschool
  @text << " #" + "codeschool"
end
```

```
tweet.hash_codeschool
tweet.hash_codeschool
```

calls method_missing

calls hash_codeschool