



University of
Sheffield

COM1001 *Introduction to Software Engineering* SPRING
SEMESTER

Ruby Blocks

Professor Phil McMinn

You may think that **Sinatra** and **RSpec** involve special Ruby **keywords** that you've not seen before.

Like “**get**” (**Sinatra**) or “**describe**”, “**context**” and “**it**” (**RSpec**):

```
require "sinatra"

get "/hello-world" do
  "Hello, World!"
end
```

```
describe "#string_comparison" do
  context "when given two strings, 'aardvark' and 'aargh!'" do
    it "returns 3" do
      expect(string_comparison("aardvark", "aargh!")).to eq(3)
    end
  end
end

# ...
```

The truth is these are not special keywords at all – they're actually **methods**.

```
require "sinatra"

get "/hello-world" do
  "Hello, World!"
end
```

```
describe "#string_comparison" do
  context "when given two strings, 'aardvark' and 'aargh!'" do
    it "returns 3" do
      expect(string_comparison("aardvark", "aargh!")).to eq(3)
    end
  end
end
# ...
```

The truth is these are not special keywords at all – they’re actually **methods** of those frameworks.

But how can this be, given that we can nest lines of code within do/end statements, like `if` statements, for example?

It’s because they use a special feature of Ruby, called **Blocks**.

Understanding Blocks

The & indicates the parameter is a block. A method can have up to one block parameter. Like a regular parameter, the block parameter can have any name – it just has to start with &. The block parameter must always be last in the parameter list.

This is the block of code we're passing into the method. The method can be called with any arbitrary sequence of statements, as many times as we like.

This method accepts a block of code as a parameter – "&block".

```
def my_block(&block)
  puts "Hi!"
  block.call
  puts "That's all folks!"
end

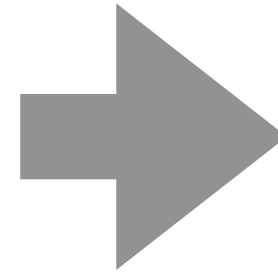
my_block do
  puts "> This is being executed within"
  puts "> the my_block method"
end
```

ruby/block.rb

The Output

```
def my_block(&block)
  puts "Hi!"
  block.call
  puts "That's all folks!"
end

my_block do
  puts "> This is being executed within"
  puts "> the my_block method"
end
```



```
codio@north-mister:~/workspace/com1001-code/blocks$ ruby block.rb
Hi!
> This is being executed within
> the my_block method
That's all folks!
```

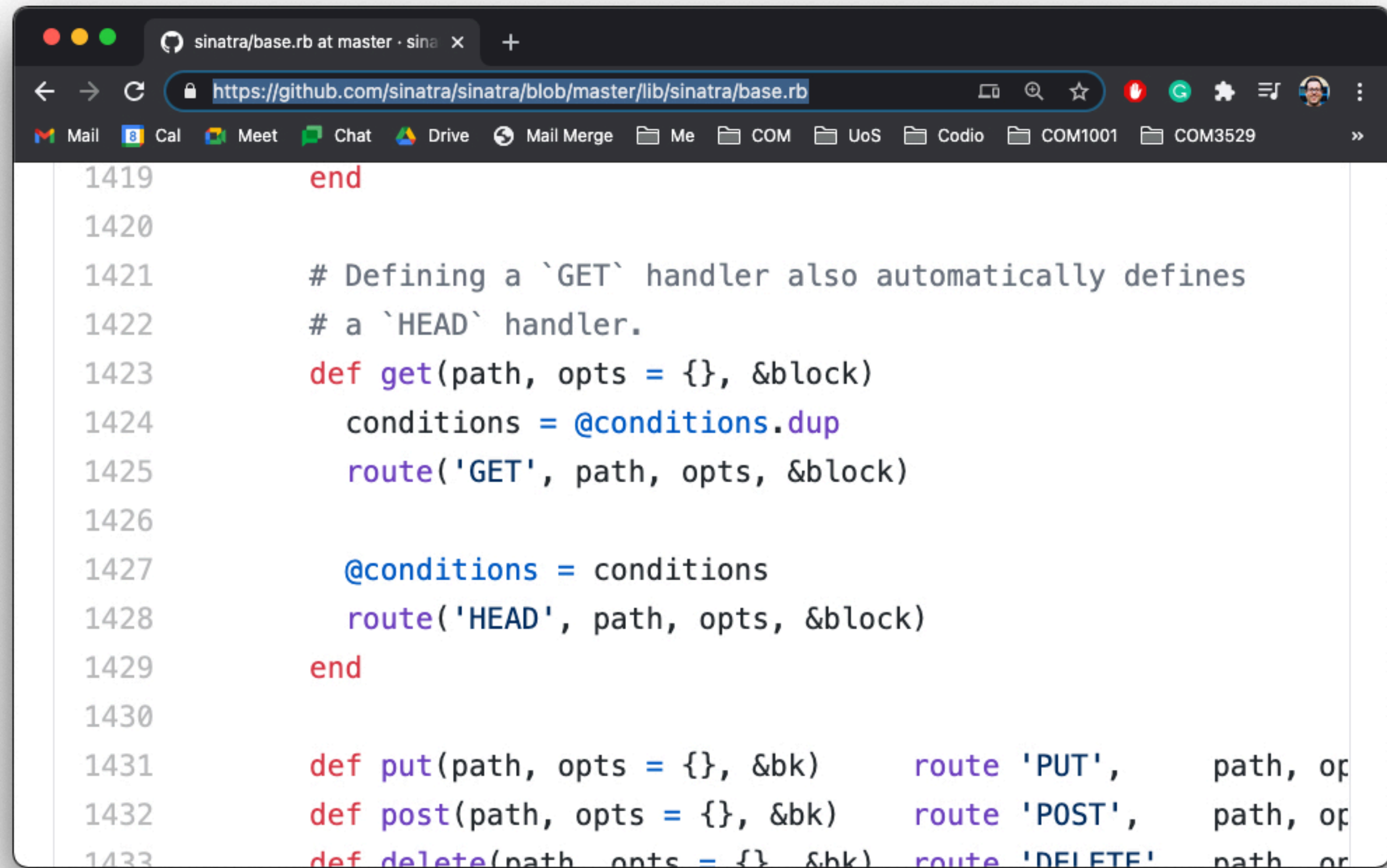
Domain Specific Languages

Blocks make it easy to construct **Domain Specific Languages (DSL)** with Ruby.

A DSL is a programming language specialised to a particular application domain, for example:

- Web Application Programming – **Sinatra**
- Testing – **RSpec**

Blocks make it look like we're using a “special” language, when actually they're just being passed into regular Ruby methods.



The screenshot shows a web browser window with the address bar displaying `https://github.com/sinatra/sinatra/blob/master/lib/sinatra/base.rb`. The browser's tab is labeled "sinatra/base.rb at master · sina". The browser's address bar and tabs are visible at the top. Below the browser window, the source code for the `get` method is displayed. The code is as follows:

```
1419     end
1420
1421     # Defining a `GET` handler also automatically defines
1422     # a `HEAD` handler.
1423     def get(path, opts = {}, &block)
1424       conditions = @conditions.dup
1425       route('GET', path, opts, &block)
1426
1427       @conditions = conditions
1428       route('HEAD', path, opts, &block)
1429     end
1430
1431     def put(path, opts = {}, &bk)      route 'PUT',      path, op
1432     def post(path, opts = {}, &bk)    route 'POST',    path, op
1433     def delete(path, opts = {}, &bk)  route 'DELETE'  path, or
```

Here's the definition of the `get` method for Sinatra, from GitHub...

There is a lot more to blocks that we have covered here. For example:

- Blocks can have parameters
- You can call methods with blocks, even without a block parameter
- Blocks belong to a category of programming language machinery called **lambda expressions**.

Lambda expressions feature in many programming languages and can be implemented and may be utilised in those languages in a variety of different ways.

For more information about blocks and lambda expressions in Ruby, take a look at:

<https://www.rubyguides.com/2016/02/ruby-procs-and-lambdas/>