COM 1001

Introduction to Software Engineering

Professor Phil McMinn

Ruby Blocks

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**):

```ruby
require "sinatra"

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

```ruby
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
  # ...
```

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

```ruby
require "sinatra"

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

```ruby
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
  # ...
```

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 method accepts a block of code as a parameter – “&block”.

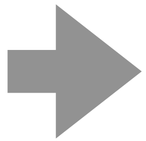
This is the block of code we’re passing into the method. The method can be called with any arbitrary sequence of statements, as many times as we like.

```ruby
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

```ruby
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!
```

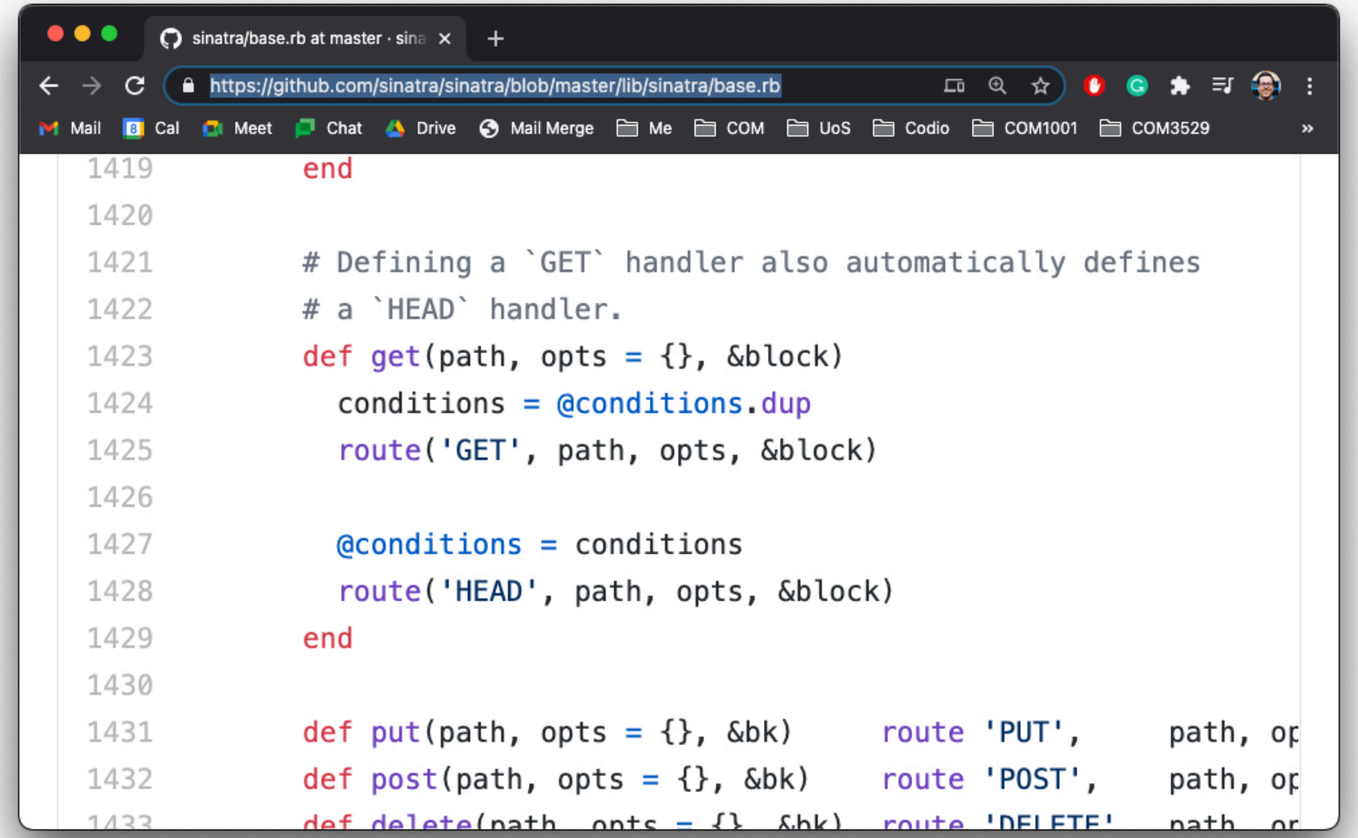
Domain Specific Languages

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.



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 are implemented and may be utilised in those language 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/`