University of Sheffield

COM1001 SPRING SEMESTER
Professor Phil McMinn
p.mcminn@sheffield.ac.uk

Queries

How can web pages take inputs?

So far, there has been no actual need to generate any of our HTTP responses dynamically – all of what we have seen could have been done statically.

That is, **none of our routes have responded to inputs**.

How can we supply inputs to web pages?

One way is by adding information to the end of URL in the form of a **query**.

Queries

Queries are an optional part of the end of a URL that start with a question mark, `?`

A query consists of key-value strings appearing in the form `key=value`, separated by ampersands, `&`, e.g.:

```
https://someurl.com/?firstname=Phil&surname=McMinn
```

This **query** contains two key value pairs:

`firstname`:`Phil`, and `surname`:`McMinn`

The `params` hash

Sinatra puts the key-value strings of a query into variable called `params` that we can access in a Sinatra block for a route.

This is one way in which web pages in Sinatra can handle inputs.

The `params` variable is a type of data structure in Ruby referred to as a **hash**. **A hash stores key-value pairs**, so it is ideal for storing the information in a query.

A hash is similar to a `dict` **in Python** and a `HashMap` **in Java** (if you happened to have encountered them).

Hashes in Ruby

You can think of a hash as a two column table, where the first column is for each the key, and second column is for the value corresponding to the key.

Hashes come up quite frequently in Ruby/Sinatra programming.

The key must be unique. If a key-value pair is entered into the table that has the same key as a pair already in the table, **the old key-value pair is overwritten with the new:**

my_hash

key	value
“firstname”	“Phil”
“surname”	“McMinn”

key	value
“firstname”	“Phil”
“surname”	“Foden”

output

```ruby
my_hash = {"firstname" => "Phil", "surname" => "McMinn"}

puts my_hash["firstname"] + " " + my_hash["surname"]

my_hash["surname"] = "Foden"

puts my_hash["firstname"] + " " + my_hash["surname"]
```

“Phil McMinn”

“Phil Foden”

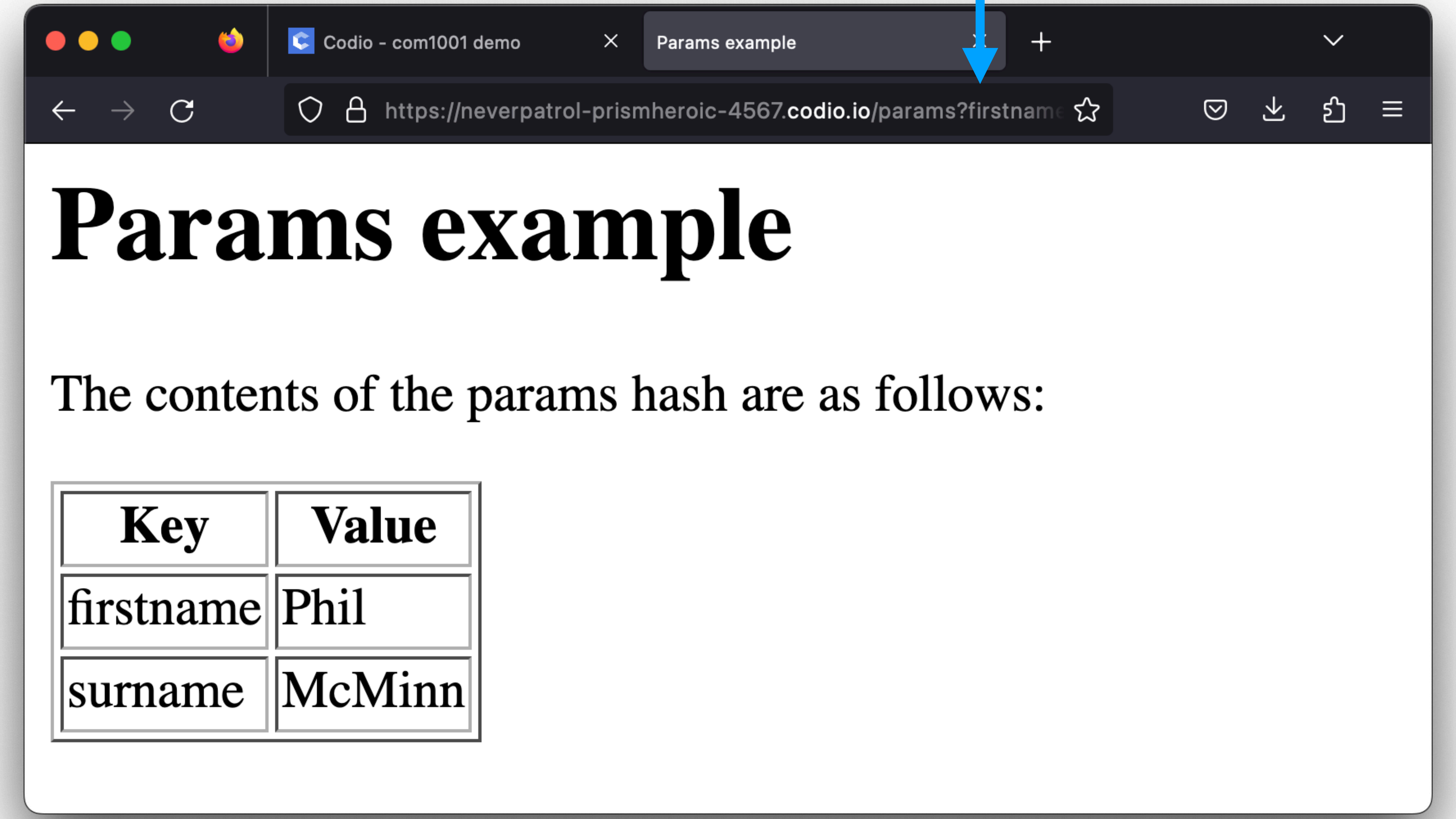
Queries and the params hash

This params_example in the GitHub repository returns a string of the contents of the params hash as passed in through the query.

```
<html>
<head>
  <title>Params example</title>
</head>
<body>
  <h1>Params example</h1>
  <p>The contents of the params hash are as follows:</p>
  <table border="1">
    <tr>
      <th>Key</th>
      <th>Value</th>
    </tr>
    <% params.each do |key, value| %>
    <tr>
      <td><%= key %></td>
      <td><%= value %></td>
    </tr>
    <% end %>
  </table>
</body>
</html>
```

week1/params-example/views/params.erb

https:// [...] /params?firstname=Phil&surname=McMinn


Codio - com1001 demo
Params example
https://neverpatrol-prismheroic-4567.codio.io/params?firstname

Params example

The contents of the params hash are as follows:

Key	Value
firstname	Phil
surname	McMinn

Live Demonstration:

`week1/times_table_with_params_example`

(from the COM1001 GitHub repository)

Featuring:

- Use of a query with the `params` hash
- Validating values passed to a route from the `params` hash

Crafting a Query String

```
require "uri"

get "/construct-querystring" do
  person = { "name" => "Phil McMinn",
             "job" => "Professor of Software Engineering",
             "address" => "Regent Court",
             "age" => "That would be telling..." }

  @querystring = URI.encode_www_form(person)

  erb :construct_querystring
end
```

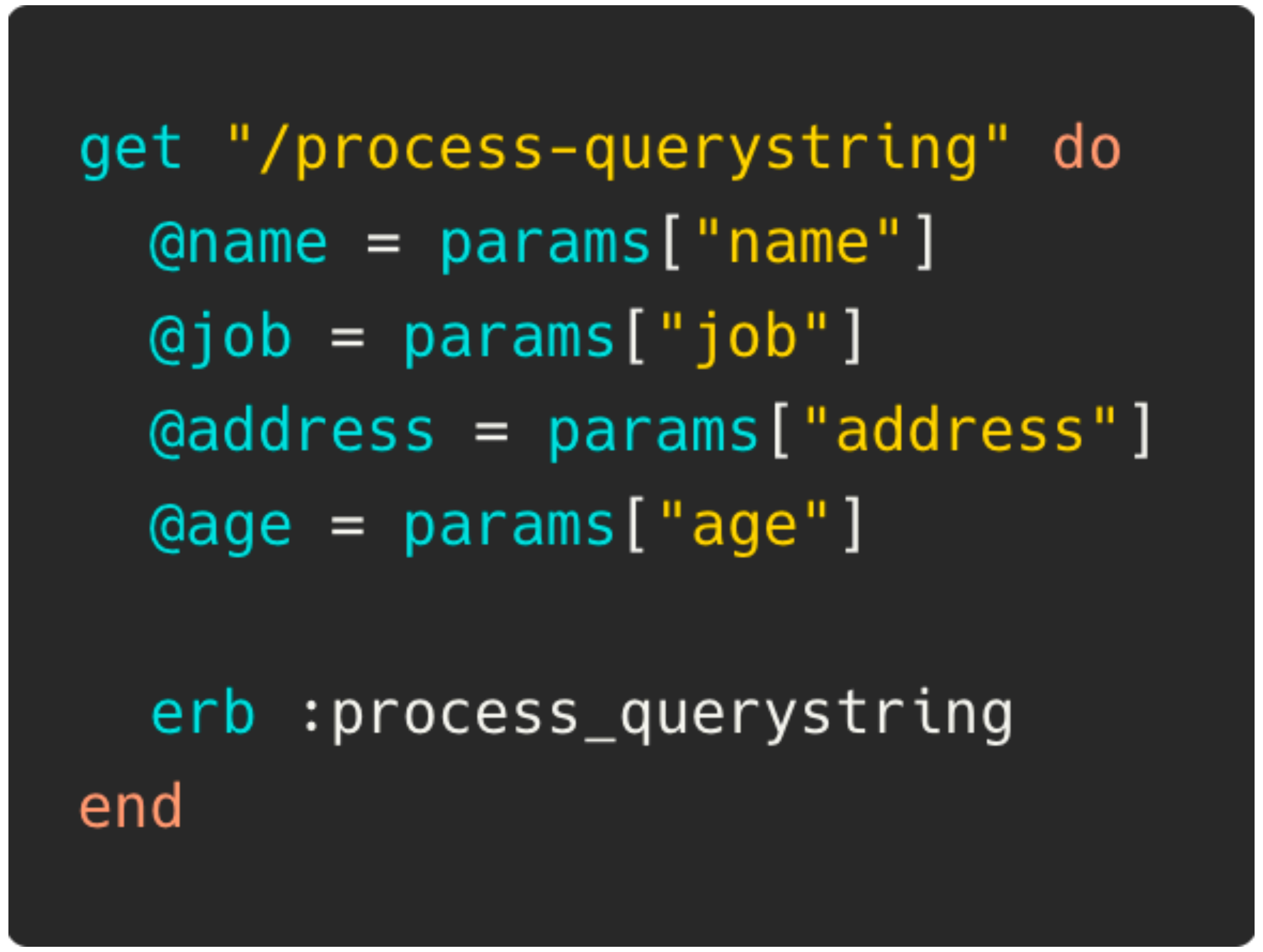
Sometimes it's useful to pass values from one page to another via query in the URLs in `<a href="…">` tags. We can turn any arbitrary hash into a query string using the `URI.encode_www_form` method

```
<a href="/process-querystring?<%= h @querystring %>">Click me!</a>
```

We always need to escape the query string in the view, since "&" (the key-value separator in a query string) is a HTML special character. This will only work properly if we've not already escaped any key-value pairs.

The value of `@querystring` is:

`/process-querystring?name=Phil+McMinn&job=Professor+of+Software+Engineering&address=Regent+Court&age=That+would+be+telling…`



```ruby
get "/process-querystring" do
  @name = params["name"]
  @job = params["job"]
  @address = params["address"]
  @age = params["age"]

  erb :process_querystring
end
```

`forms/simple_forms/controllers/querystring.rb`
(part 2/2)

The route corresponding to the URL we sent the query string with can then unpack the `params` hash and get the values we originally sent.

Note this code is no different to that had the values had been submitted by a form – the receiving route has no idea of the context, or how the data got in the `params` hash – just that it is there!

Queries – Summary

Queries are a series of key-value strings added to the end of a URL following a question mark, `?`

After matching a route from the URL, Sinatra will parse a query and put the key-value strings into a Ruby hash data structure called `params`.

The `params` hash can be accessed in the route's block and views.