University of Sheffield

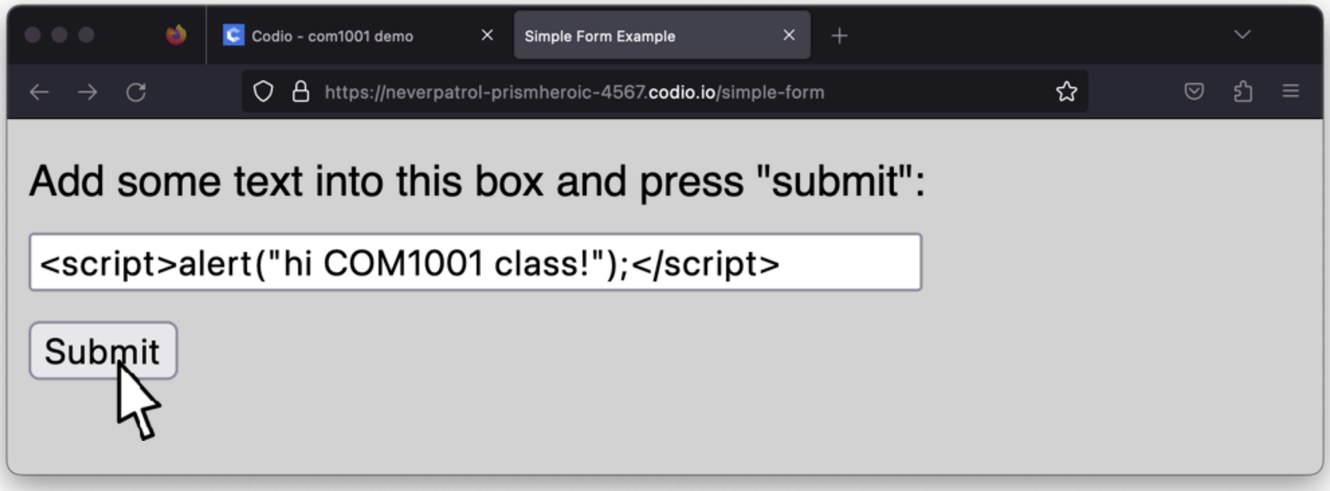
COM1001 SPRING SEMESTER

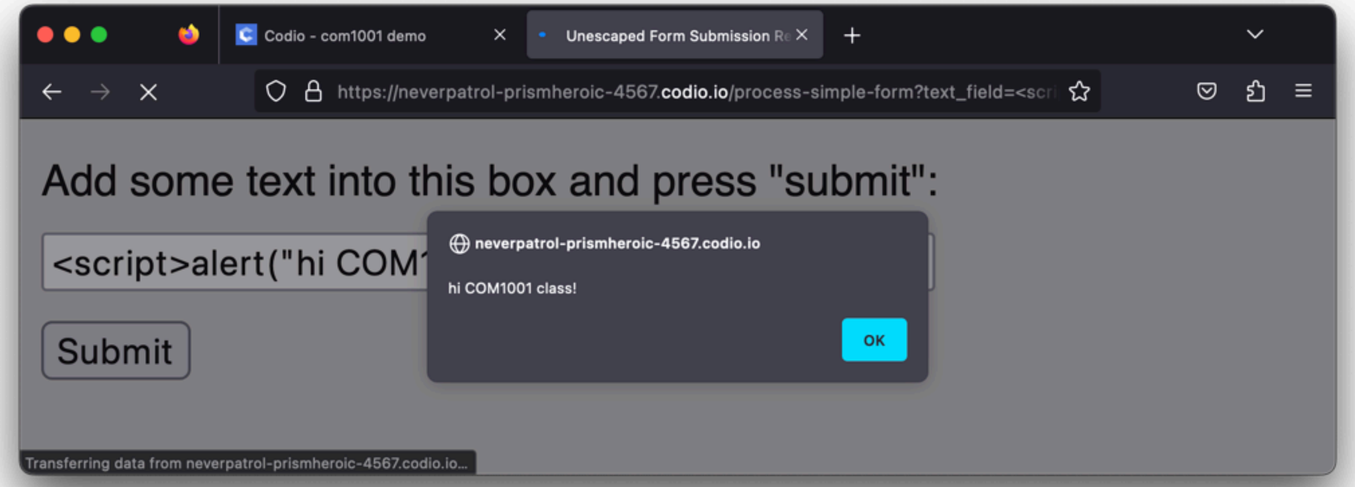
Professor Phil McMinn

p.mcminn@sheffield.ac.uk

Sanitising and Validating User Inputs

Never Trust User Inputs!





What is Happening?

Let’s Investigate the HTML Source of the Page

The text the user submitted is injected directly into the page and the browser has no way to distinguish it not page of the regular HTML.

As such, the browser dutifully runs the JavaScript in the `<script>` tags, launching an alert dialog, effectively hijacking our page!

How to view the source of the generated page:

Chrome: View → Developer → View Source

Firefox: Tools → Browser Tools → Page Source

Safari: Settings →Advanced → check that “*Show features for web developers*” is enabled; then Develop → Show Page Source

Others: Google it!

What Can We Do?

We need to avoid using certain characters that could be interpreted as being part of the HTML code of the page – but how?

HTML defines a set of **HTML entities** for so-called **HTML special characters** such as these, including angled brackets `<` and `>`.

HTML entities start with an `&` and end with a `;`. **The original character can be replaced with its equivalent HTML entity and the Browser will then know it's part of the content of the page, not the HTML formatting itself.**

For example `<` can be replaced with `<` and `>` can be replaced with `>`

So `<hello>` becomes `<hello>`

This process of this sub-sequence replacement to ensure strings are interpreted correctly is often referred to as "**escaping**" the string. More generally, the process of making user inputs safe for use is called **sanitising inputs**.

Luckily Ruby has built in methods to do escaping for us, with minimum fuss.

```erb
<html>
  <head>
    <title>Escaped Form Submission Result</title>
    <link rel="stylesheet" href="style/style.css">
  </head>
  <body>
    <% unless @submitted_text_field_value.nil? %>
      <p>You submitted the text:<strong>
      <%= h @submitted_text_field_value %>
      </strong></p>
    <% end %>
  </body>
</html>
```

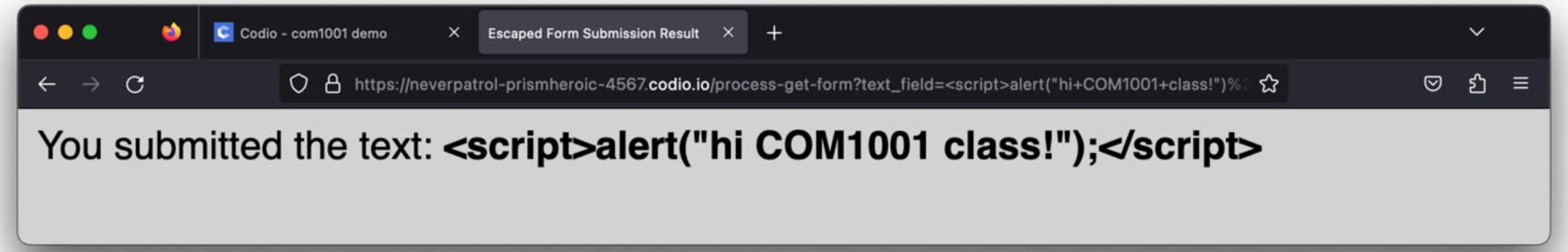
Sanitisation with Escaping

We can escape HTML special characters using the `h` method.

(Need the line `include ERB::Util` in our main `app.rb` file to use this)

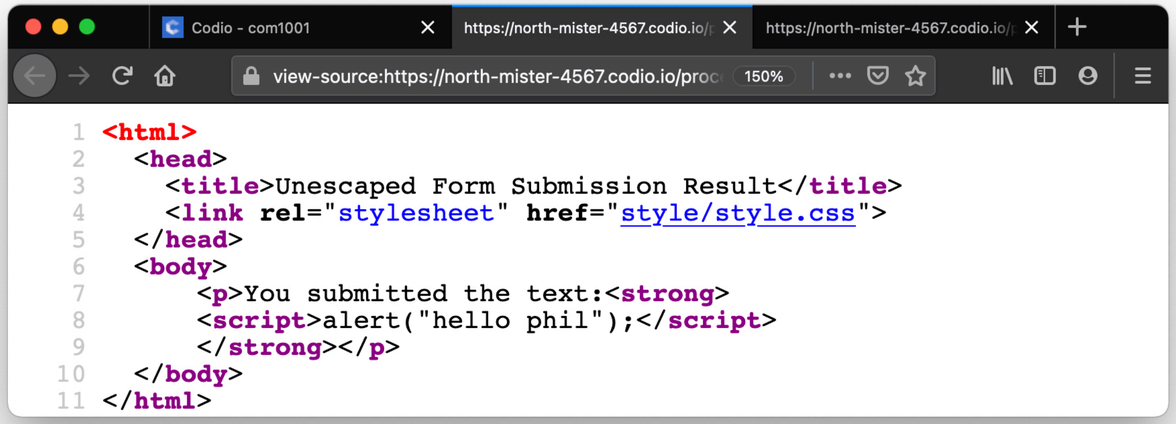
This will prevent the browser interpreting any user inputs as part of the HTML of the page, such as maliciously injecting scripts and potentially compromising the security of our web application.

Result:



The Difference in the Page Source

Unescaped (view not using h):




```
<html>
  <head>
    <title>Unescaped Form Submission Result</title>
    <link rel="stylesheet" href="style/style.css">
  </head>
  <body>
      <p>You submitted the text:<strong>
      <script>alert("hello phil");</script>
      </strong></p>
  </body>
</html>
```

Escaped (view uses h):

```
<html>
  <head>
    <title>Escaped Form Submission Result</title>
    <link rel="stylesheet" href="style/style.css">
  </head>
  <body>
      <p>You submitted the text:<strong>
      &lt;script&gt;alert(&quot;hello phil&quot;);&lt;/script&gt;
      </strong></p>
  </body>
</html>
```

Special HTML characters like “<”and “>” are replaced with equivalent HTML “entities” (`<` and `>` respectively) that will render those characters as text, so as not to confuse the browser into thinking they are part of the HTML of the page.

Other Types of Sanitisation

There are other types of data cleansing we might want to perform, depending on the application.

For example, **removing leading and trailing whitespace from a user's form entry**.

Extra spaces at the end of an entry for a person's name should not be counted as part of their name.

Nor are they part of an email address, or a telephone number.

Validation

We also need to validate user inputs.

Validation means checking the user entered the right kind of data before the application does any processing on it.

Some examples:

Important information is not missing, like the user left their name blank.

Numbers entered into forms are actually numbers.

Email addresses are in the right format

Recall the Times Table Example

The example took two strings from the query of the URL that needed to be numbers.

Not validating they were actually numbers before doing multiplication would have caused the application to **crash**.

```ruby
get "/times-table" do
  # set @multiplicand if the "m" key is set in the params and its an integer
  param = params.fetch("m", nil)
  @multiplicand = param.to_i if str_digits?(param)

  # set @limit via the same process
  param = params.fetch("l", nil)
  @limit = param.to_i if str_digits?(param)

  # if we have values for @multiplicand and @limit at this point (that is,
  # they are not nil) then we have valid parameters (else, we do not)
  @params_valid = !@multiplicand.nil? && !@limit.nil?

  erb :times_table
end
```

times_table_with_params/controllers/times_table.rb

1. Check if string is made up of digits, i.e., constitutes a positive integer

2. Convert string to integer

str_digits?

`str_digits?` is not part of the standard Ruby library

It actually lives in a module called Validation found in `helpers/validation.rb`

This module contains a series of functions useful for validating strings in forms, including `str_email_address?`, `str_uk_telephone?`, and others.

`helpers/validation.rb` is included by `app.rb` so the functions in it are available to all Sinatra apps created using `create_app`

What are Helpers?

`validation.rb` is an example of a helper module.

Helper modules provide functions that the rest of the application can easily use, e.g. to validate a string or format a string.

Helpers live the `helpers` directory of the application.

To add your own, add them as a module to the helpers directory. Make a file called `local_helpers.rb` and use `include` to include them. Ask if you need help with this!

Live Demonstration:

the `santisation_validation` example

(from the com1001-examples GitHub repository)

Featuring:

- Validation and sanitisation of different form field types
- Helpers and adding a new helper