The Other Art of Computer Programming. Milestone 5: HTML & CSS. 1990s

Melanie Tarr

Edith Cowan University, melanie.tarr@gmail.com

Follow this and additional works at: https://ro.ecu.edu.au/ecuworkspost2013

Part of the Education Commons, and the Programming Languages and Compilers Commons

10.25958/5e3a76e662fd9
This Book is posted at Research Online.
https://ro.ecu.edu.au/ecuworkspost2013/7440
Learn HTML and cascading style sheet scripting for the world wide web.

Form submission and receival
Learn how to get and send information.

The Ruby Framework
Learn how to make a simple ‘todo’ list.

1990s

Milestone

HTML & CSS

The Other Art of Computer Programming
by Melanie Tarr
1990

1990 Tim Berners-Lee set up a communication between a web browser and web server. This started Hypertext Transfer Protocol for the Internet.
Table of Contents

Instructions........................................................3
Client, server and browser......................................4
Universal Resource Locators..................................5
Model View Controller..........................................6
Simple HTML and CSS...........................................7
MVC and web apps...............................................10
View and form.....................................................11
CSS elements.....................................................10
CSS inheritance ..................................................11
Website file structure..........................................12
Tags.......................................................................13
What does index.html look like.............................14
The syntax of CSS................................................16
How forms work when submit is pressed...............22
Ruby on rails framework.......................................24
Welcome the user................................................26
Adding an item to the database.............................27
Back to the controller...........................................30
Testing your Ruby application..............................31

Note: This lesson may offer useful explanations for
the DT curriculum codes above and elaborations in
the Australian National Curriculum.
The lesson progresses from simple to complex web systems and is divided into three stages. The first stage is HTML and CSS, the second explains how web services work, and the final stage explains how web servers work. The Ruby framework is a complex web service however it still contains HTML and CSS from the first stage of the lesson. The third part of the lesson is explained using a software pattern called model view controller.
THE WORLD WIDE WEB OR MORE SIMPLY THE WEB - IS A GLOBAL SYSTEM OF LINKED DOCUMENTS ACCESSED THROUGH THE INTERNET, WHICH IS ITSELF A GLOBAL COMPUTER NETWORK.

THE WEB USES A CLIENT-SERVER MODEL THOUGH WHICH WEB PAGES ARE RETRIEVED FROM WEB SERVERS AND THEN VIEWED IN SOFTWARE APPLICATIONS RUNNING ON THE CLIENT COMPUTER, CALLED WEB BROWSERS.
Web pages are located using an addressing system called a Uniform Resource Locator or URL. A URL looks like this:

http://theotherart.com/books

This URL consists of three parts.

1 The beginning of the URL (http://) is the scheme or protocol. The HTTP protocol stands for Hypertext Transfer Protocol (HTTP). Other schemes you are likely to encounter include HTTPS, FTP and SSH.

2 The middle part of the URL, theotherart.com specifies the server where this resource resides.

3 The last section (/books) identifies the specific resource on this server.
Model View Controller (MVC) is a software pattern that has three interconnected parts. This pattern is popular in web applications. A web application is software that is designed to work on the web.

One of the first MVC applications was trialled in the Smalltalk language.

The individual components of MVC are model, view and controller. The MVC pattern manages the data, logic and rules of the application.
THE CONTENT AND STRUCTURE OF THE WEBPAGE IS HTML WHILE WHAT THIS ACTUALLY LOOKS LIKE IS THE STYLE OF THE CONTENT OR CSS

Simple html and css

FOR EXAMPLE:

```
#body {
    font-size: 12em;
    description of element
description of element
}
```
THE HTML IS THE STRUCTURE OR THE MODEL WHILE THE CSS IS THE VIEW OF THE MVC PATTERN

MVC and web applications
The view: form

The web has both a form and a style that are connected. For example, the form of the image to the right is a “p” while the style of the form is as follows:

Font-size: 72 pixels high
Colour: grey

Control over the form of the letters is limited to set characteristics.
A HIERARCHICAL REPRESENTATION OF THE WEBPAGE IS TO THE RIGHT. EACH ELEMENT CAN ALSO HAVE ITS OWN HEADING STYLE. THE LOWER STYLES INHERIT FROM THE UPPER STYLES. FOR EXAMPLE, A HEADING IN THE FOOTER WILL INHERIT THE DETAILS IN THE ELEMENT (SKIN) ABOVE IT.
CSS inheritance

```css
#sidebar h3, #footer h3 {
  font-family: Serif;
  font-size: medium;
  line-height: 100%;
  color: black;
  text-shadow: 0px 2px 3px #555;
}
```

This style description indicates a deviation from the existing h3 values. In this case the footer element inherits its form from the sidebar.

```
#footer h3 {
  font-size 1.5em;
  text-shadow: 1px 1ps 4px #111
}
```

As we want to alter the form of heading three in the footer, we override the styles the footer inherits from the sidebar heading three value. The following style appends or overrides the existing values.
The website file structure

This lesson started with CSS because it is the look or the form of the information displayed on a webpage. The CSS is also the view component of the Model View Controller pattern. In contrast, the HTML is the structure or the model of the webpage. The default name for the homepage is index.html.

The file structure of a website has a specific order of files. These files form a structure like the user interface to identify and locate files. Regardless of what graphical user interface you use, the website structure is usually similar to the one above.
The tags specify the linear order of the HTML file.

The tags within the text are in pairs made up of beginning and ending tags. For example, a beginning HTML tag or `<html>` will have an ending HTML tag or `</html>`. These tags must match within the file or the computer will not understand how to structure the internet page.

1. `<html>`
   - This tag specifies that the page is an HTML page.

2. `<head>`
   - This tag specifies the header section on the HTML page.

3. `<title>`
   - This tag specifies the title section of the HTML page.

4. `<body>`
   - This tag specifies the body section of the HTML page.
IF YOU WERE TO LOOK AT THIS PAGE IN THE BROWSER IT WOULD NOT LOOK LIKE A NORMAL WEBPAGE BECAUSE THE HTML HAS NOT BEEN LINKED TO THE CSS FILE. THEREFORE THE CONTENT HAS NOT BEEN STYLED OR FORMATTED

Viewing the html file now in a browser
TO CONNECT THE HTML FILE TO THE STYLE OF ITS ELEMENTS, WE SPECIFY THE FOLLOWING:

```
<link rel="stylesheet" type="text/css" href="style.css" />
```

WHAT RELATIONSHIP THE CASCADING STYLE SHEET HAS TO THE HTML FILE.

THE TYPE OF FILE

LOCATION AND NAME OF THE CSS FILE SO THAT THE WEBSERVER KNOWS WHERE TO FIND THE FILE.

THESE THREE FEATURES ARE ENCODED INTO THE HEAD SECTION OF THE HTML FILE.
CSS styles are short lists of the styling rules of each element in the index.html file. The styling rules are stored in the style.css file.
The syntax of CSS

**The list of the rules of the header element are stored in the styles.css file**

The header element is a simple and straightforward element of CSS to implement within HTML, other elements are not so straightforward.

```css
#header {
  background-color: olive;
  color: white;
  padding: 10px;
}
```
The syntax of css

THE BODY TAG IS THE NEXT SECTION IN styles.css

THE BODY ELEMENT ENCAPSULATES THE ENTIRE PAGE OF THE WEBSITE. IT HAS A WHITE BACKGROUND

body {
    background-color: white;
}

THE FIRST LINE SPECIFIES THE BACKGROUND COLOUR AS OLIVE

THE SECOND LINE LISTS THE TEXT AS BEING WHITE IN COLOUR
The syntax of CSS

The container division is the yellow box on the page and is placed behind other page elements like the header.

#container

```css
#container {
  background-color: yellow;
  width: 800px;
  margin-left: auto;
  margin-right: auto;
}
```

The first line specifies the background colour as yellow.

The container element position should be under the body tag description in the CSS file.

The next three lines include the auto code that centres the division or <div> horizontally.
The syntax of css

The content, navigation and main divisions are examined on this page.

By clearing the float elements, a division is justified against the parent container that the child style has inherited elements from.

The navigation floats to the left of its parent division.

The footer needs to be cleared of both float elements.
The syntax of css

THE footer DIVISION IS EXAMINED ON THIS PAGE

That is all the elements you need to describe in style.css for a styled webpage.

```css
footer {
  clear: both;
  padding: 10px;
  background-color: purple;
  text-align: right;
}
```
The syntax of css: in summary

To describe the features of a tag within a HTML file, list the tag name and enclose the description within curly brackets.

```css
#header {

}

#navigation {

}

#main {

}
```
How forms work when “submit” is pressed

The form is submitted, that is, the submit button is pressed on a webpage by a person.

A request to the server is sent. The request is called a get request.

The server understands the request is dynamic and sends the data to the web app for processing.

Different types of HTTP patterns are sent to the server to be processed by the web application software.

- POST
- GET
- DELETE
- PUT
- PATCH
- HEAD

Depending on the URL received, the server determines how this is handled.

The URL pattern determines how the server handles the request.

These requests are often made to the database for a specified field like a user.

For example, get "users" would return a list of all entries in the field user in the database.

A get request asks the web server to retrieve data which may be housed in a database.

...
How forms work when “submit” is pressed

The web app analyses the request from the pattern in the URL.

Then the web app matches data in the submitted form with information in its database.

The HTML page is created with information in the database. The database information goes into placeholders in the HTML.

Welcome to the database!
Please enter your details

First name
Last name
Email
Phone

Submit

The browser processes the HTML structure returned from the server. Any CSS or JavaScript is processed now.

Any static files are loaded and displayed in the browser.
Ruby on rails framework

WEB SERVERS STORE AND PROCESS THE INFORMATION DELIVERED TO CLIENTS

THE INFORMATION IS MOST OFTEN DELIVERED IN THE FORM OF A WEB PAGE

AS THESE SYSTEMS BECAME MORE COMPLEX, FRAMEWORKS WERE INTRODUCED TO MANAGE THE COMPLEXITY

What are GEMS?

GEMS MANAGE THE PACKAGES IN RUBY THAT PROVIDE A STANDARD FORMAT FOR DISTRIBUTING RUBY PROGRAMS AND LIBRARIES. THE "GEM" IS A FORMAT CONTAINING THE TOOL THAT MANAGES THE INSTALLATION AND SERVER FOR DISTRIBUTION OF THE GEMS

What is a web framework

RUBY ON RAILS IS AN EXCEEDINGLY POPULAR FRAMEWORK USED BY PEOPLE WANTING TO PUBLISH THEIR OWN WEBSITES

RAILS REMOVES THE NEED FOR THE PROGRAMMER TO KNOW HOW TO PROGRAM AND MANIPULATE THE DATABASE WITH A LANGUAGE LIKE SQL

WEB APPLICATION DEVELOPMENT HAS BEEN INFLUENCED BY RUBY WITH FEATURES SUCH AS SEEMLESS DATABASE TABLE CREATIONS, MIGRATIONS AND VIEW SCAFFOLDING

RUBY IS CHARACTERISED BY HAVING LESS SEMICOLONS, BRACKETS AND LOOKS LIKE MORE OF A STRUCTURED ENGLISH THAN A PROGRAMMING LANGUAGE

MVC web architecture

MANY WEB FRAMEWORKS ARE BASED ON THE MVC SOFTWARE PATTERN

THE MODEL VIEW CONTROLLER PATTERN SEPARATES THE CODE INTO EACH OF THE THREE COMPONENTS

THE SEPARATION IS KNOWN AS MODULARISATION

Model
FUNCTION: Storage managing and organising
EXAMPLE: Databases
EXAMPLE: Data files

View
FUNCTION: Presentation
EXAMPLE: User interface
EXAMPLE: Html
EXAMPLE: Some PHP

Controller
FUNCTION: Handles passing information from the view to the model
EXAMPLE: PHP
a) Welcome the user (the program)

The next stage in the lesson, Stage 3, describes the Ruby on Rails framework and implementation of a web application called to do list. The app creates a list of items and adds them to a database.

A web framework can be overwhelming to look at because of the amount of files generated. However, the code for the design pattern, model view controller (MVC) in these folders is all you need to look at!

The command:

```
rails new todolist
```

Will generate the above directory structure.

```
http://www.todolist.com
```

Welcome to the todolist app!

As someone arrives at the home page by typing the path into the browser: `https://www.todolist.com`

They will see a welcome page.

Within the controller folder you should have a `welcome_controller`.

Using HTML language navigate to the `index.html.erb` and type in a welcome message:

```
<h1>Welcome to the todolist app!!</h1>
```

Programming in Rails generally follows the order below:

- The program
- The routes
- Controllers
- Views
- Models
b) Configure the routes

Routing is how execution navigates through all the different files in MVC and orders the execution of those files. When the user opens a webpage, we need configure the welcome index controllers method for the routes. The routes tell the program what to do next and which file to go to.

The following instructions specify the order of execution of the framework files, i.e. configure the routes of the web app. The order is as follows:

- The user invokes the welcome controller by going to a site (HTTPS location)
- The welcome controller is triggered (welcome_controller) and invokes the Ruby index file which is a view the user sees
- Edit config/routes.rb by uncommenting the line root 'welcome#index'

Test:
Start your rails server and visit the home page to test the web application.

When you use Ruby there are four edits to the files:
- Generate the framework;
- Generate the controller;
- Generate the index;
- Create a route.
WE HAVE A VIEW AND CONTROLLER COMPONENTS BUT WE DON'T YET HAVE ANY MODEL. RUBY ON RAILS TAKES CARE OF ALL THE SQL (STRUCTURED QUERY LANGUAGE) A PROGRAMMER NEEDS TO UNDERSTAND TO MAKE THE WEB APPLICATION WORK. RAILS INTERACTS WITH THE SQL (STRUCTURED QUERY LANGUAGE) DATABASE VIA STRUCTURED ENGLISH

THE PROGRAM THE ROUTES CONTROLLERS VIEWS MODELS

specify the data types you would like in the model. Name columns to represent data in the database. These columns form the model in MVC.

class CreateItems <ActiveRecordMigration> def change create_table : items t.string : title t.text : notes t.timestamps end end

THE RAILS FRAMEWORK DEALS WITH ALL THE SQL (STRUCTURED QUERY LANGUAGE) LANGUAGE DATABASE. YOU DON'T NEED TO PROGRAM ANY SQL (STRUCTURED QUERY LANGUAGE). RUBYgenerates the code to the left

39
Adding an item to the database

THE PATH (ROUTE) IN THE WEB PAGE BELOW (https://todolist/items/new) GENERATES A NEW PAGE FOR AN ITEM. IT ALSO MEANS CREATE A FORM FOR A NEW ITEM

DATA IS ENTERED BY THE USER AND THEN SAVED BY CLICKING ON THE "SAVE ITEM" BUTTON

THE PROGRAM THE ROUTES CONTROLLERS VIEWS MODELS

THEIR ARE NEW AND CREATE METHODS WITHIN THE ITEMS CONTROLLER FILE:
- def new REQUIRES THE PAGE
- def create POSTS THE INFO

ONCE THE ROUTE IS TRIGGERED THE CONTROLLER GENERATES THE VIEW FILE

SQL

RUBY CODE WITHIN THE VIEW FILE TELLS THE FORM WHERE TO GO AFTER SUBMITTING THE ITEMS_PATH

ONCE THE SAVE ITEM BUTTON IS CLICKED, THE DATA CONCERNING THE NEW ITEM IS SENT TO THE DATABASE.
Back to the controller

**THE TABLES AND HAVE BEEN CREATED BUT THE FRAMEWORK DOES NOT KNOW WHAT TO DO WITH THE TABLES. THE CONTROLLER PUTS DATA INTO THE DATABASE AND TAKES ITEMS OUT**

```
def index
  @items = Item.all
end
```

**CREATE A GLOBAL VARIABLE TO CONTAIN AN ARRAY OF ITEMS OR A LIST**

```
def show
  @items = Item.find(params[:id])
end

SHOW ALL THE ITEMS IN THE LIST ON THE VIEW
```

```
def create
  @items = Item.new(item_params)
  @items.save
  redirect_to @items
end

CREATE A NEW ITEM
```

```
def new
  @items = Item.new
end

MAKE A NEW ITEM
```

```
def destroy
  @item = Item.destroy(params[:id])
  redirect_to items_path
end

DELETE THE ITEM
```

```
def edit
  @items = Item.find(params[:id])
end

EDIT THE ITEM
```
NOW TRY ACCESSING YOUR APPLICATION IN THE BROWSER. ALL THE ROUTES AND VIEWS NEED TO BE TESTED WITHIN THE RUBY APPLICATION

Testing your Ruby application