Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

CSU44000 Internet Applications

Week 1 Lecture 2

Conor Sheedy

HTTP Overview

Client Server model

- a request–response protocol
- Client requests, server responds

Application layer protocol

- Assumes a reliable transport layer protocol
- Usually, Transmission Control Protocol (TCP)
  - QUIC, new transmission layer protocol used by HTTP/3

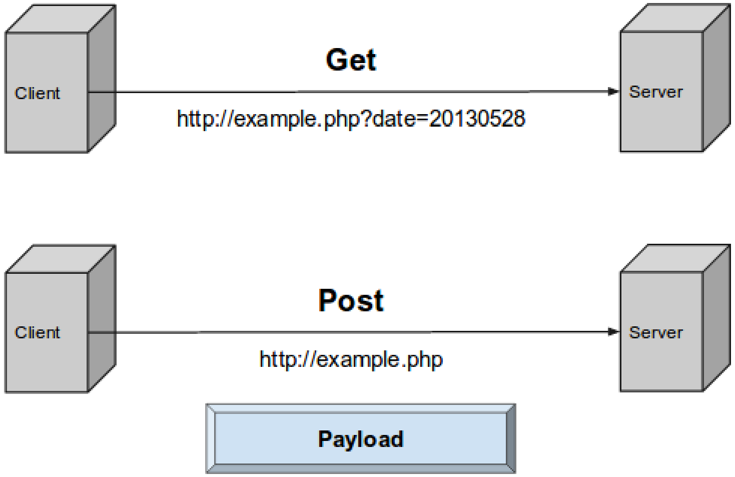
HTTP is:

- Connectionless
- Media Independent
- Stateless

HTTP resources identified using Uniform Resource Locators (URLs)

- URLs are encoded as hyperlinks in HTML document

Simple HTTP exchanges



Form Data, JSON Strings, Query Parameters, View States, etc

HTTP Request

HTTP Messages

- Request
- Response
- Always have a start line
- Sometime have headers
- Sometimes have a body

Request Method (Verb)

Responses

- Status Code
- Extensible
- Starting with a number:
  1. Informational
  2. Success
  3. Redirection
     - Further Action Required
  4. Client Error
  5. Server Error

HTTP Request Messages ‘Verbs’

GET

- Requests resource
- Client wants to retrieve data only

POST

- Client send data to the resource

PUT, DELETE, CONNECT, OPTIONS, TRACE, PATCH

- Perform other useful actions
- most important 2 are GET and POST

HTTP Request Messages ‘Verbs’

The less useful ones

PUT

– replaces all current representations of the target resource with the request payload

DELETE

– deletes the specified resource

CONNECT

– establishes a tunnel to the server identified by the target resource

OPTIONS

– describes the communication options for the target resource

TRACE

– performs a message loop-back test along the path to the target resource

PATCH

– applies partial modifications to a resource

HTTP with HTML, what next?

perfectly sufficient for navigating a hypertext document (Web 1.0)

- but later people wanted to use the web for everything
- The client was widely used
- There was a simple protocol that was very widely implemented
- There were many servers deployed

Maybe instead of writing dedicated applications that run on the server end

we could use this as ....

the Universal Program Interface

Concept of the Web as the Universal Program Interface

People wanted to use the web for navigating more complex data than static HTML files.

- Culminated in web applications like HoTMaiL (1996)
    - a startup bought by Microsoft for 450 million in 1997
    - reading your email on the browser was revolutionary
- Google 1998, went public in 2004
- Blogger 1999 , bought by Google 2003
- Wikipedia 2001
- Picasa 2002, bought by Google 2004
- Oddpost 2002, first webmail using Ajax
- Gmail launched 2004, early use of Ajax (Asynchronous JavaScript and XML)
- Google Maps 2005

Modern Internet Applications

- **Web 2.0**
  - User generated content
  - More interactive

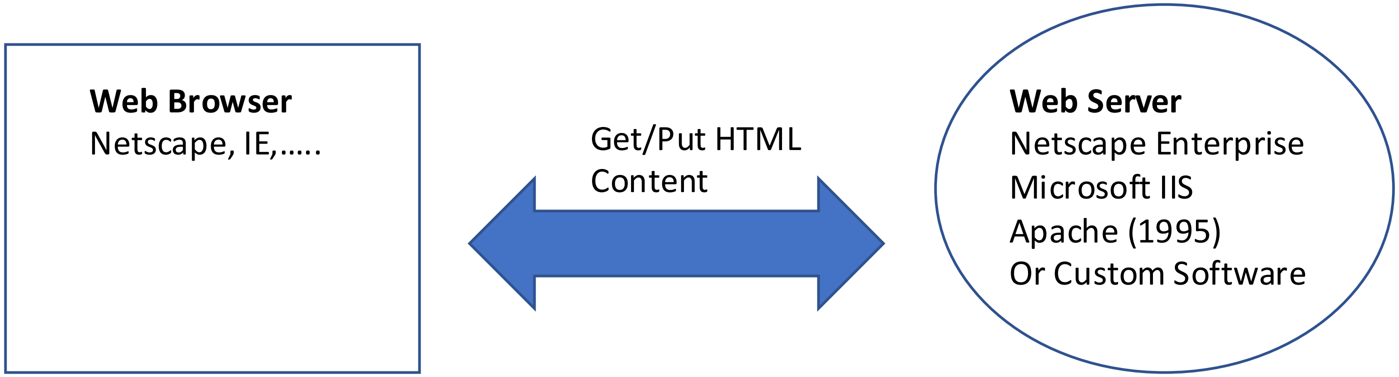
Where to put the logic?

Could do this by building logic into the

WebServer (Server) or the

WebBrowser (Client Side)

or a combination of BOTH



Web Browser
Netscape, IE,…..

Get/Put HTML
Content

Web Server
Netscape Enterprise
Microsoft IIS
Apache (1995)
Or Custom Software

Making the Web Browser do Clever Things

Early Hypertext was aimed at representing the 'Structure' of the Document – not the presentation

The browser was in charge of rendering the doc – possibly with 'hints' in the HTML e.g. <i>Text to be put in Italics</i>

Later this changed with 'Styles' and Cascading Style Sheets (CSS)

- Css was released at the end of 1996
- Css style sheets allow separation of presentation and content
- multiple web pages can share formatting

Selection of Server-Side Technologies

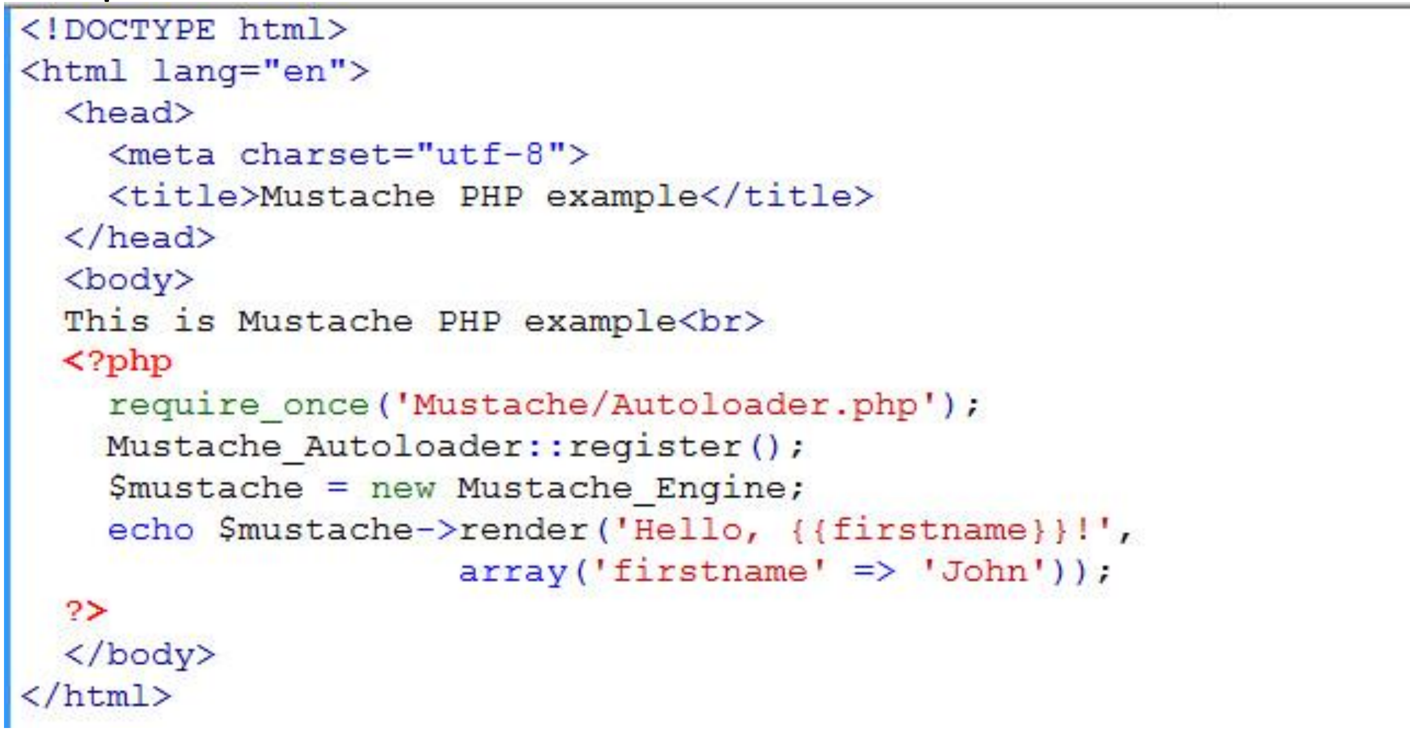
Common Gateway Interface – CGI-Bin (NCSA, 1993)

- Html Form data sent with
- Http request
  - (e.g. GET with parameters appended to URL)
  - POST with payload
- With a URL for the script
- The server launches the script (written in Perl, C, or another scripting language)
- The script consumes the GET or POST parameters
- and generates HTML for the response

Selection of Server-Side Technologies

PHP (Personal Home Page 1995)

- Used the CGI-BIN in conjunction with server-side scripts mixed with the HTML to generate Dynamic HTML pages
- PHP Hypertext Preprocessor, 1997 rename



```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Mustache PHP example</title>
  </head>
  <body>
  This is Mustache PHP example<br>
  <?php
    require_once('Mustache/Autoloader.php');
    Mustache_Autoloader::register();
    $mustache = new Mustache_Engine;
    echo $mustache->render('Hello, {{firstname}}!',
                    array('firstname' => 'John'));
  ?>
  </body>
</html>
```

Selection of Server-Side Technologies

Web Frameworks

- Many based on the Model–view–controller (MVC) architectural pattern
  - Separates Data model, from user interface and controller
- three-tier organisation
  - Client
  - Application (on the server)
  - database

Ruby on Rails (2005)

- Ruby is the language
- Rails is a model–view–controller (MVC) framework
  - default structures for
    - a database, a web service, web pages
- Open Source

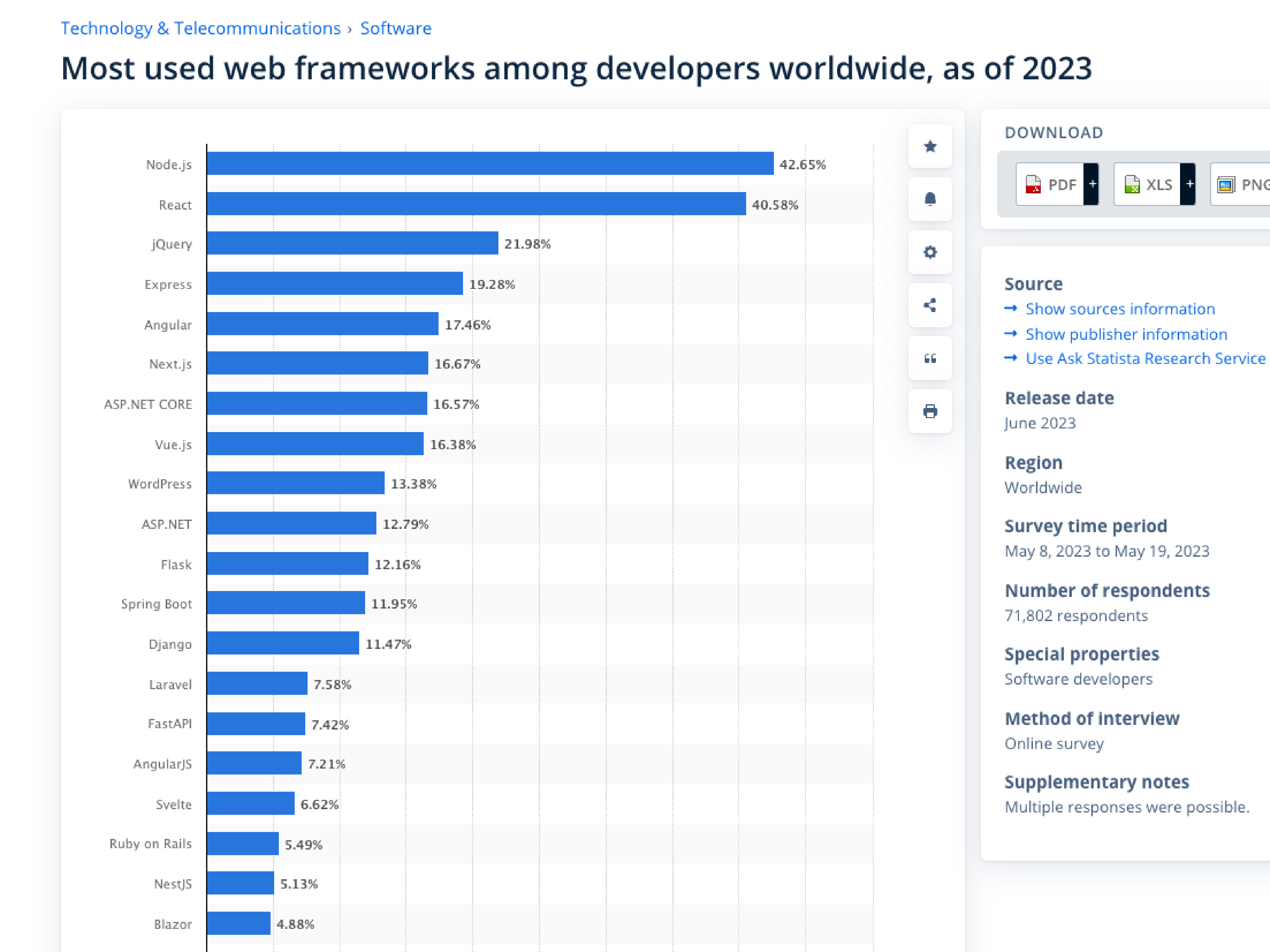
Selection of Server-Side Technologies

Other Web Frameworks

- Grails
- Django

Node.JS (2009)

- back-end JavaScript runtime environment
- Uses V8 JavaScript Engine
- Open Source
- Facilitates use of JavaScript on the server-side
- Npm package manager
- use an API designed to reduce the complexity of writing server applications
- Node.js functions are non-blocking, event-driven programming



Technology & Telecommunications › Software

Most used web frameworks among developers worldwide, as of 2023

Source

→ Show sources information

→ Show publisher information

→ Use Ask Statista Research Service

Release date

June 2023

Region

Worldwide

Survey time period

May 8, 2023 to May 19, 2023

Number of respondents

71,802 respondents

Special properties

Software developers

Method of interview

Online survey

Supplementary notes

Multiple responses were possible.

Quiz: Given a choice of doing work on the server side or client side – which is best for:

Responsiveness?

Security?

Efficiency?

Carbon Footprint?

Proprietary Control of the Service?

1. **Server-Side**

2. **Client Side**

Browser Evolution

Mosaic

- developed by NCSA (Marc Andreesen & Eric Bina) for Windows in 1993
- 3 years after 'WorldWideWeb' browser

Netscape Navigator, 1994

- Andreesen teamed up with Jim Clark (SGI founder)
- free for non-commercial use

Microsoft Internet Explorer, August 1995

- May, 1995: Bill Gates sends his "Internet Tidal Wave" memo to MS staff
- free and bundled with windows
- quickly became dominant
- bundling was unfair competition
- Antitrust case settled in 2001

Browser Evolution

Browser Wars began

- – IE won!
- [Visualization of Browser Wars: https://www.youtube.com/watch?v=es9DNe0l0Qo

Netscape became Mozilla

- which later gave birth to Chrome
- Google chrome made open source
  - Chromium in 2008
  - contained V8 Scripting engine
  - Microsoft based Edge on Chromium, Opera, Brave too

Scripting in the Browser

Netscape's Brendan Eich

- co-founder of Mozilla, now CEO of Brave
- was given 10 days to produce a scripting language for Navigator
- Released 'LiveScript' in Dec'95
- called 'JavaScript' in Dec'95 release
- rumour was he wanted to use Scheme but he dismisses this.

In August'96 Microsoft put VBScript into Internet Explorer

- modelled on Visual Basic
- The engine also included Jscript
- based on Netscape's JavaScript

Numerous incompatibilities arose during Browser Wars

- There was a need for standardisation

Scripting in the Browser…

Netscape submitted JavaScript to ECMA in November 96

ECMAScript, June 97

- ES2(1998)
- ES3(1999),
- ES4 was controversial,
- ES5.. (ES6 later renamed ECMAScript 2015),
- ECMAScript 2016
- Currently on ECMAScript 2023 (ES14)

Scripting in the Browser…

Language is evolving (being enlarged!) all the time

- ES.Next
  - Finished proposals
  - Will be in the next standard
- Requires 'learning to learn' to keep up to date
- This article from 2016 talks about many commonplace features which were cutting edge at the time gives an excellent feel for what it can be like to develop with tools that continue to evolve

https://hackernoon.com/how-it-feels-to-learn-javascript-in-2016-d3a717dd577f

Scripting in the Browser…

TypeScript

- Microsoft released in 2012, Open source
- Superset of JavaScript with optional **static typing**
- JavaScript is not a typed language
- Many people like discipline and checking of assigning types
  - Fewer bugs in large projects
- use a transpiler to translate in into JavaScript
- but typing/classes have not settled yet

CoffeeScript

- ‘nicer’ syntax
- Compiles to JavaScript

Scripting in the Browser…

Where is JavaScript today?

- When features enter the ECMAScript standard they have to implemented by Browsers
- This link tracks which features of the language have been implemented in Node.js
- http://node.green
- New standards are usually a superset of the old with extra features

Javascript Basics

Syntax of language:

- Blocks of code, variables similar to Java
- but basic language is dynamically typed
  - no type checking
- Can use ‘Strict’ mode to force all variables to be declared
- No input-output as part of the language
  - Can use the console, but the output doesn’t appear on the page

JavaScript runs in a ‘Host Environment’ with Objects (host specific) available to:

– Get and Put output text, results, colours, sounds etc

– Events and Actions that JavaScript can ‘Hook on to’

Simple Javascript Program running in Node.JS

Syntax similar to Java

- “use strict”;
  - Forces you to declare all your variables
  - This is good practice
- **let i=0;**
  - ‘var’ has global scope
    - (outside of a function)
    - Redeclaration doesn’t cause an error
    - A good example of fixing a rapidly designed language
      - ES2015 (ES6)
  - Variable declaration
  - Has block scope

```
"use strict";
console.log("Watch while i count to 10");

let i=0;
while (i<= 10)
{ console.log("Counting %d", i);
i++}
console.log('finished');
```

Simple Javascript Program running in Node.JS

```
"use strict";
console.log("Watch while i count to 10");

let i=0;
while (i<= 10)
{ console.log("Counting %d", i);
i++}
console.log('finished');
```

- **Console object** provides access to the browser's debugging console
  - allows text string to be
    logged and will appear in the console

Everything is dynamically typed with no checking

Early versions of JavaScript the code was interpreted

- Now JIT (Just-in-time) compilation is often used
  - To make it faster

Similar Program running in a browser

JS can be mixed with HTML enclosed with <script> tags

- Browser feeds contents to its built in JavaScript engine, (e.g. V8 in Chrome)

In this host, JavaScript can use objects to access:

- The browser windows, frames, tabs
- The debug console
- It also has Read/Write access to the Document Object Model (DOM) which completely controls what is shown to the user and How it is presented

```
<html>
<head>Page used to execute simple script</head>
<body>
<H1>I'm Busy - check my console</H1>
</body></h1>
</html>

<script>
"use strict";
console.log("Watch while i count to 10");

let i=0;
while (i<= 10)
{ console.log("Counting %d", i);
i++}
console.log('finished');
window.open(); //open new tab
</script>
```