Web Development

Portland State
Computer Science

Web development (at a 10k level)

- Typical web application components
  - Programming language
  - Framework
  - Template system

- Rendering approaches

Programming language

Compiled or Interpreted

- Compiled
  - To machine code
    - No run-time needed, fast
    - Good for IoT devices with limited resources
  - To bytecode
    - Requires run-time VM environment to execute
  - Longer development cycle
  - Inability to patch quickly
    - Must recompile entire app (Apache Struts bug and Equifax)
    - Dev and Ops involved to fix security flaws (versus just Ops)
- Interpreted
  - Scripting languages
  - Requires interpreter and all packages application depends upon to be present
  - Slow, but some languages with good support for JIT
    - Python (PyPy), Javascript (v8, NodeJS)
    - Performance closer to compiled – often single threaded

Static vs. Dynamic types

- Static
  - Types checked at compile-time
  - Type errors caught at compilation *before* deployment
    - Debug then deploy
  - Good for mission (and business) critical applications
- Dynamic
  - Types checked at run-time

```
var variable = 3;
variable = foo(0);
typeof(variable);  // ?
```

```
function foo(somenum) {
    if (somenum == 3)
        return '0';
    else
        return 0;
}
```

  - Type errors caught at run-time via crashes (Python, JavaScript)
    - Or not caught at all via generation of nonsensical results with type coercion (PHP)
  - Overhead
    - Type checking must be done for each use (compared to static typing that does not require checks)
  - Deploy then debug, good for rapid prototyping

Strongly vs. Weakly typed

- Strongly typed
  - Requires explicit type conversion

```
$ python -c "print '5' + 8"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```

- Weakly typed
  - Implicit type conversion & casting
  - Type coercion that *automatically* changes a value from one type to another
    - PHP

```
$ cat math.php
<?php print('5' + 8); ?>
$ php math.php
13
```

- C coercion
  - What does this output?

```
#include <stdio.h>
int main() {
  char c=0x80;
  printf("%x\n",c);
}
```

```
mashimaro <~>  1:24PM % ./a.out
ffffff80
```

But only on x86/Linux since char is unsigned for C on ARM

- Javascript coercion

```
'5' - '2' ==    // 3
'5' + '2' ==    // '52'
5 - '2'   ==    // 3
5 + '2'   ==    // '52'
'5' - 2   ==    // 3
'5' + 2   ==    // '52'
```

- Example (along with arcane rules for Javascript coercion with ==)

`0 == "0"`

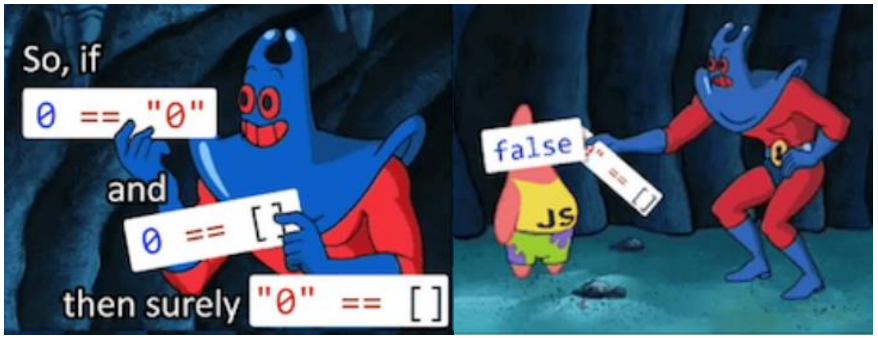
- *True.*
  - *If x is Number and y is String, return x == ToNumber(y)*

`0 == []`

- *True*
  - *If x is String or Number and y is Object, return x == ToPrimitive(y)*
    - *where ToPrimitive coerces to String, Number or Boolean*
  - *Empty array is an object. Objects are first coerced via .toString()*
  - *Returns an empty string.*
  - *Then, empty string coerced via ToNumber() for comparison, returns 0*

`"0" == []`

- *False. Leads to hilarious memes*



  - https://www.destroyallsoftware.com/talks/wat
  - https://www.freecodecamp.org/news/explaining-the-best-javascript-meme-i-have-ever-seen/

Type inferencing

- Automatic detection of types so programmer doesn't have to declare it
  - Good for language conciseness

```
let x = 3;  <= x inferred to be a number
```

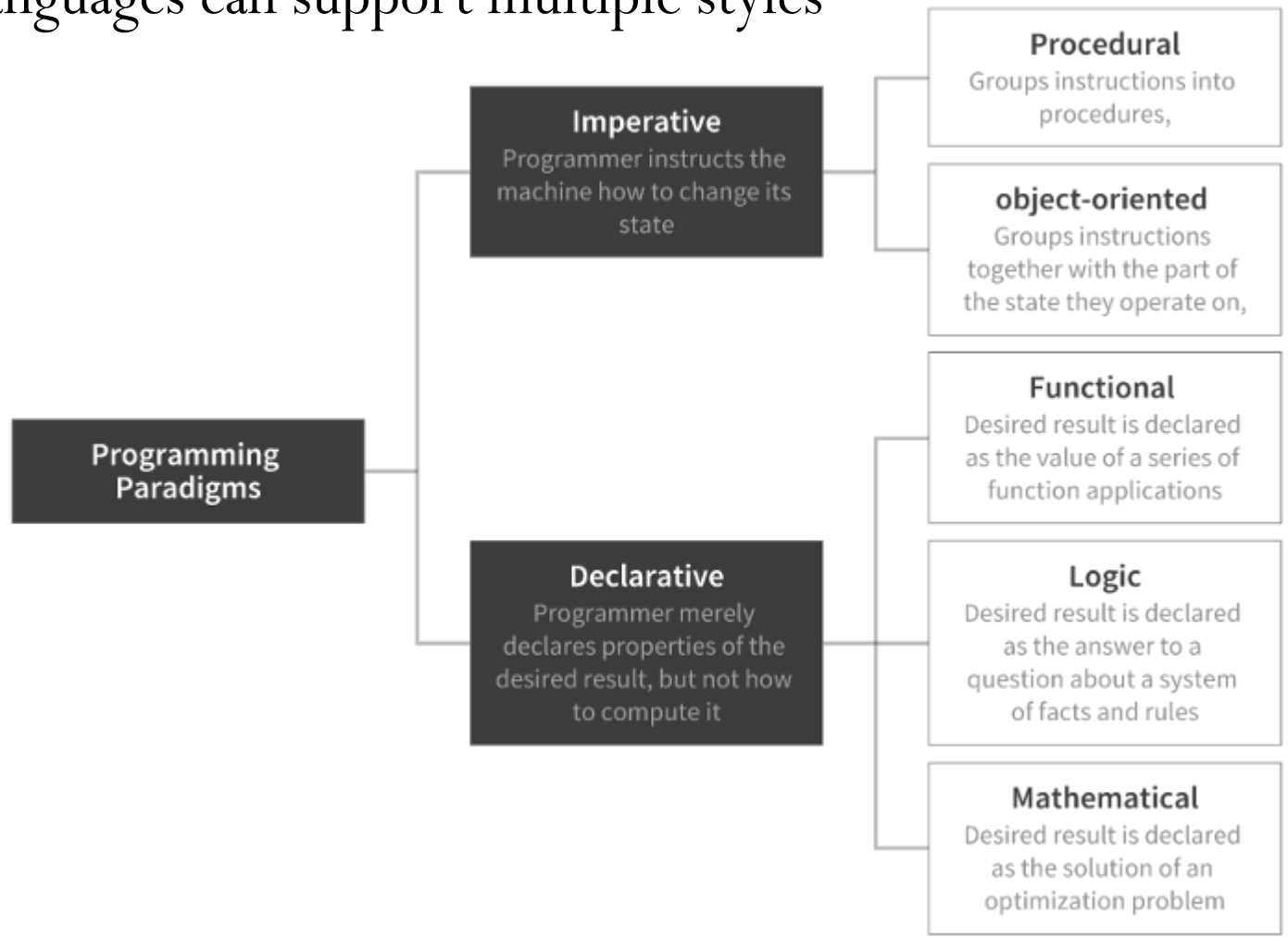
  - Kotlin vs. Java

```
// Type is ArrayList
val a = arrayListOf("Kotlin", "Scala", "Groovy")
```

  - Local variable type inferencing added to Java 10 (3/2018)

Programming paradigms supported

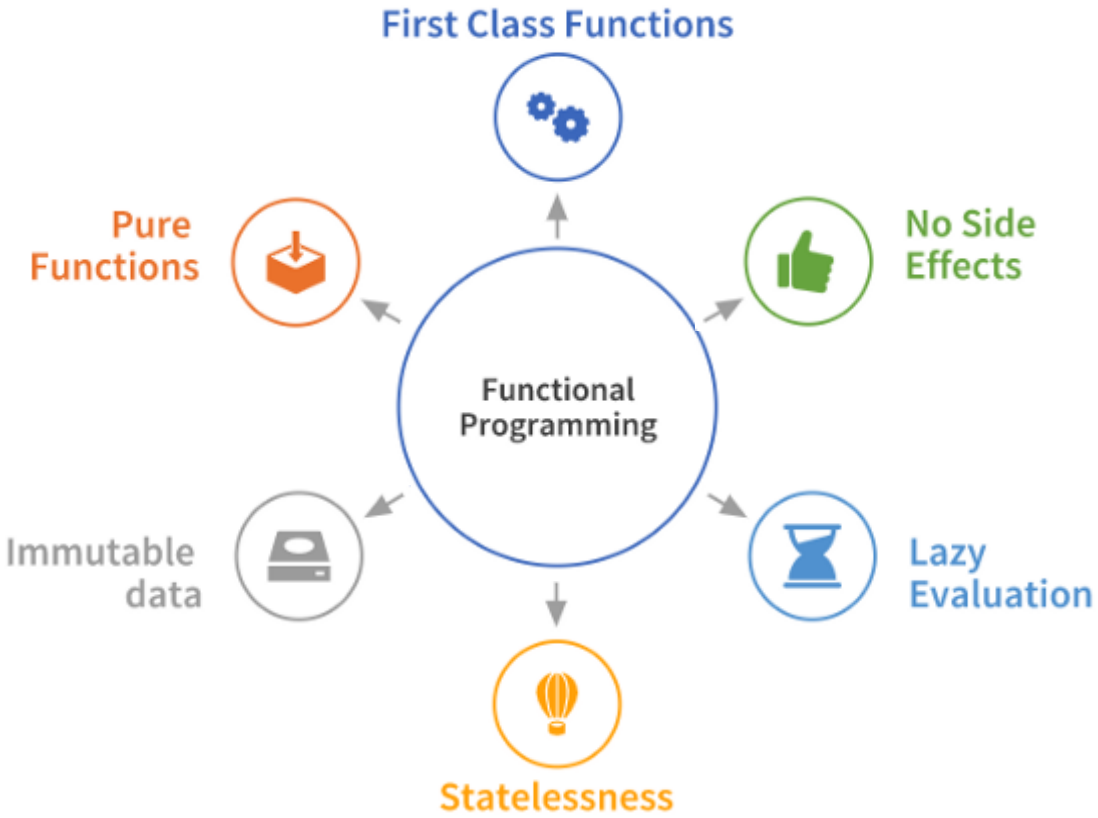
- Languages can support multiple styles



Functional programming support

- Helps manage complexity while reducing errors
  - CS 457/557
- Web frameworks moving to paradigm to manage complexity
  - React, Angular, Redux, Elm…
- Similar goal as object-oriented programming
  - "Object-oriented programming makes code understandable by encapsulating moving parts. Functional programming makes code understandable by minimizing moving parts." Michael Feathers

- Computation as evaluation of mathematical functions
  - Functions define what to do rather than perform an action
- Stateless (only depend on data passed as arguments)
- Avoid changing state and any mutable data
  - Rather than modify a list, create new copy with modifications instead
- First-class functions where functions treated as objects
- Laziness (only compute things on-demand)
- Pattern applied heavily in web development and data science
  - https://towardsdatascience.com/elements-of-functional-programming-in-python-1b295ea5bbe0



Asynchronous support

- Optimizing single-threaded operation for performance
  - Event-driven programming and non-blocking operations
  - Blocking calls automatically yield to other parts of code

Asynchronous support

- Example: Callbacks
  - Register a function to be executed upon completion of another
  - Example
    - Synchronous file writing

```
fileObject = open(file) // blocks until file opened
fileObject.write("We are writing to the file.")  // finally can write

// do the other, totally unrelated things our program does
```

    - Asynchronous file writing

```
// open asynchronously and register writeToFile callback function
fileObject = open(file, writeToFile)

// execution continues immediately
// do the other totally unrelated things that our program does

// writeToFile executed once open finishes
```

- Other common mechanisms
  - Promises (see skipped slides)
  - async, await (Python > 3.5, Javascript ES8) (see CS 495/595)
  - Key for implementing complex web front-end UIs and progressive web applications with service workers

Concurrency support

- Ability to leverage multi-core processors
- Support for parallel execution
- Memory consistency model

Ease of development

- Programming ease
- Testing ease
- Library and package management support (maturity of ecosystem)
  - e.g. how much code you *don't* have to write
- Developer base (for hiring and for answering questions)

Ease of deployment

- Migrating from development to production infrastructure (e.g. reproducibility of execution environment)
- Updating and patching software in packages
  - Can web app be patched in-place or does it require recompilation?

Web programming languages

Java, C#

- Prevalent in e-commerce, bank web sites
  - Pre-date most web frameworks that popularized scripting languages
- Compiled+Interpreted (bytecode with JIT)
- Statically and strongly typed
  - Preferred for large sites and for critical applications
- Managed memory (garbage collected)
- Asynchronous support in both (Event interface, async/await)
- Huge developer base, mature class support, adding conciseness
- But, with deployment
  - Recompilation of apps when security patches to libraries occur
  - Vendor lock-in
    - Rely on Oracle/Microsoft to keep platform libraries secure and deploy features
    - Must buy Microsoft (e.g. Azure) to run full-blown ASP.NET applications or rely on Microsoft to keep updating .NET core for Linux

JavaScript/ECMAScript (ES)

- Designed for web browsers by Brendan Eich
- Based on Java (syntax, OOP)
- Scheme in a browser (functional programming)
- Interpreted, but with fast JIT (v8)
- Dynamically typed (type checking at run-time)
- Weakly typed (type coercion)
- Managed memory (garbage collected)
- Asynchronous from the beginning for single-threaded, event-based operation (callbacks, promises, closures, async/await etc.)
- Ease of development (400k packages for Node.js, front-end and back-end share same language)
- Ease of deployment (npm package management)

- Ideal for smaller web applications requiring quick development iterations and rapid results (IMO)

- But, type coercion and weak typing leads to "loose" equality after coercion rules applied with "=="
  - Makes "==" practically useless

```
var num = 0;
var obj = new String('0');
var str = '0';
console.log(num == obj); // true
console.log(num == str); // true
console.log(obj == str); // true
```

  - How do you *really* check for equality?
    - === (equality without coercion – "strict")
    - `Object.is()` (same-value equality)

- Cheatsheet

Sameness Comparisons

x	y	==	===	Object.is
undefined	undefined	true	true	true
null	null	true	true	true
true	true	true	true	true
false	false	true	true	true
"foo"	"foo"	true	true	true
{ foo: "bar" }	x	true	true	true
0	0	true	true	true
+0	-0	true	true	false
0	false	true	false	false
""	false	true	false	false
""	0	true	false	false
"0"	0	true	false	false
"17"	17	true	false	false
[1,2]	"1,2"	true	false	false
new String("foo")	"foo"	true	false	false
null	undefined	true	false	false
null	false	false	false	false
undefined	false	false	false	false
{ foo: "bar" }	{ foo: "bar" }	false	false	false
new String("foo")	new String("foo")	false	false	false
0	null	false	false	false
0	NaN	false	false	false
"foo"	NaN	false	false	false
NaN	NaN	false	false	true

- Also, Security issues: supply-chain attacks on NodeJS npm packages
  - Sites using packages that are abandoned (no security updates)
  - Sites relying on packages taken over by malicious developer
  - Steal credit card numbers (2018)

I'm harvesting credit card numbers and passwords from your site. Here's how.

Lucky for me, we live in an age where people install npm packages like they're popping pain killers.
Your innocence warms my heart.

But I'm afraid it's perfectly possible to ship one version of your code to GitHub and a different version to npm.

  - Steal passwords (2019)

npm Pulls Malicious Package that Stole Login Passwords

By Ionut Ilascu

August 21, 2019

- What if your code relied on these packages?

Software

How one developer just broke Node, Babel and thousands of projects in 11 lines of JavaScript

Code pulled from NPM – which everyone was using

By Chris Williams, Editor in Chief 23 Mar 2016 at 01:24 167 SHARE ▼

ZDNet MENU US

Another one-line npm package breaks the JavaScript ecosystem

An update to tiny "is-promise" library impacted millions of JavaScript projects.

By Catalin Cimpanu | April 26, 2020

TypeScript

- Weak, dynamic typing leads to software engineering problems
- TypeScript (Microsoft)
  - Typed superset of JavaScript that transpiles to JavaScript
  - Statically typed objects
  - Strongly typed objects (explicit casting)
  - Checked at compile-time
  - From 1/21/2019

> Why every new web app at PayPal starts with TypeScript
>
> 
> Kent C. Dodds Follow
> Jan 21 · 9 min read

- Similar approach: PureScript

- Mechanisms
  - Scoped variable declarations via `let`
  - Immutable constant declarations via `const`
  - Names of variables followed by a : and the variable type
  - Function declarations include parameter types and return types with similar syntax
- Compiler checks to ensure type safeness

```
// {variable name}: {variable type} = {variable value}
 // const = immutable
// let = mutable
const myString: string = 'Hello World';
let myArray1: Array<number> = [1, 2, 3];
let myArray2: number[] = [4, 5, 6]
function square(x: number): number {
  return x*x;
}
```

- Try at http://www.typescriptlang.org/play

Go (2009)

- Want the best of C/C++
  - Low-level systems programming
  - Bare-metal performance
- Want the best of Java, Python, Ruby, JavaScript
  - Managed, garbage-collected memory
  - Rich package/module support
- Want the best of JavaScript
  - First-class support for asynchrony/concurrency

Go language design

- Designed for simplicity and readability
  - "Simplicity is Complicated" (Rob Pike)
    - https://www.youtube.com/watch?v=rFejpH_tAHM
  - Features fixed (not trying to be like other languages)
    - Can fit language spec in your head (25 keywords)
    - One canonical way of doing things (unlike Perl or recently Python)
    - Easy to reason about code
  - Multi-developer coding easier

Features

- Strongly and statically typed (e.g. Java)
  - Eliminates run-time type errors and type coercion errors
  - Rich built-in types (maps, slices, first-class functions, multiple return values, iterators)
- Type-inferencing to support concise syntax
- Memory safety via managed memory that is garbage collected
  - Use Rust if this bothers you!
- Rich, built-in module support instead of includes
- Standardized code formatter `gofmt`
  - No more spaces versus tabs, no style guides needed
  - Readability built-in!

- Asynchrony and concurrency
  - Non-blocking, concurrent operation with goroutines
    - Each goroutine mapped onto an underlying OS thread
    - True parallel execution (no global interpreter lock)
    - Concise syntax
  - Shared memory disallowed to support memory-safety
    - Based on research in Communicating Sequential Processes as organizing principle in parallel applications (see CS 415/515)
    - Communication done via channels (explicit IPC)
    - Can reduce the use of semaphores/mutexes which are prohibitively expensive on modern multi-core CPUs (see CS 533)

- Example

```
func HeavyComputation(ch chan int32) {
    …
    ch <- result
}

ch := make(chan int32)

// Create a new goroutine with the same address space
// and use it to run the function.  No need to create
// thread, call library to start thread.

go HeavyComputation(ch)

result := <-ch  // Blocks if not ready
```

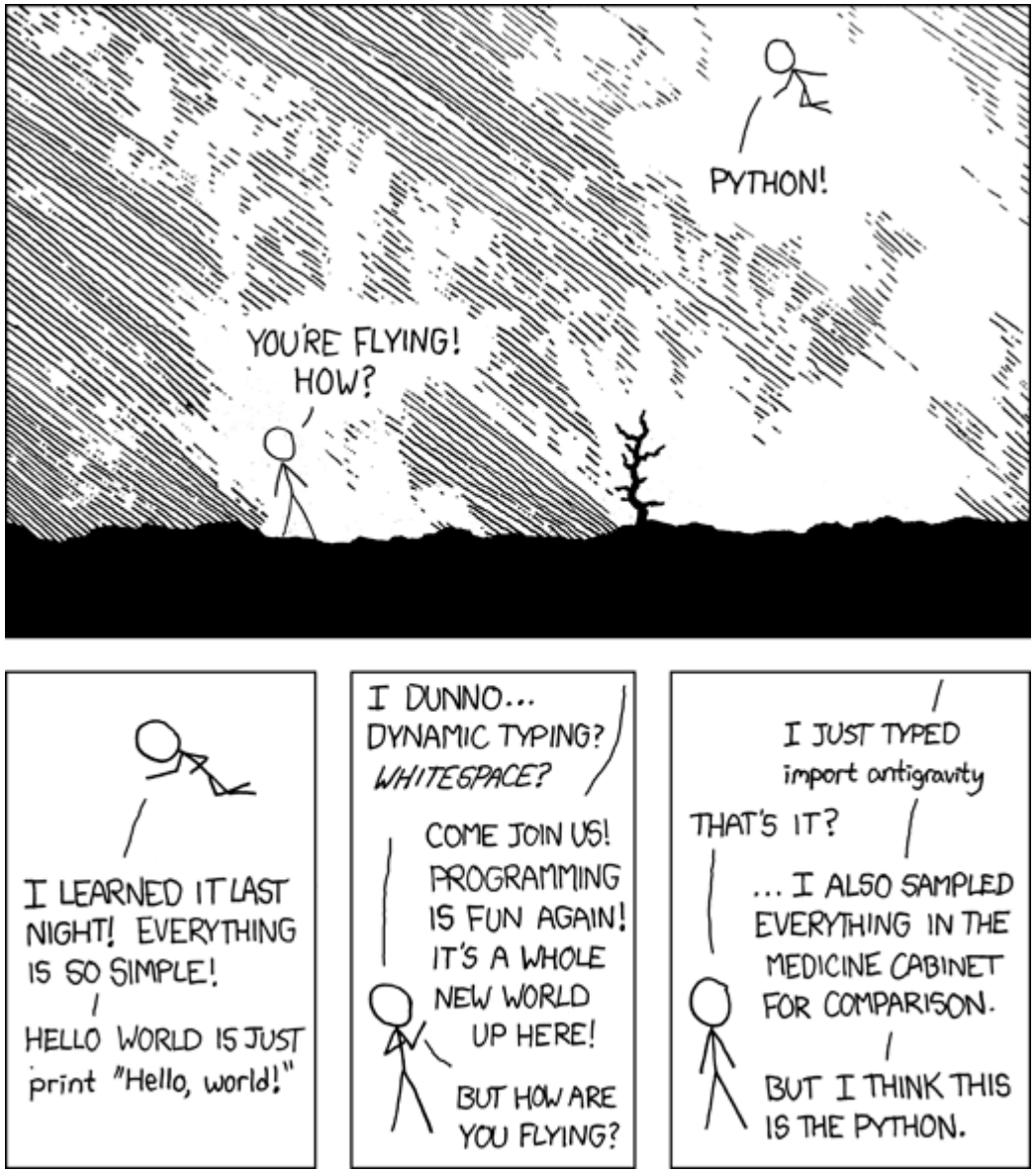
- Static, efficient compilation to bare-metal
  - Performance close to C, C++
  - "Built-in" make to avoid 45 minute compile times
- Easy to deploy
  - Emits a single portable binary with all dependencies and environment included
  - No versioning issues in deployment
  - Obviates one of the main reasons for a container
  - Used to build the world's smallest Docker container
    - https://www.youtube.com/watch?v=zu8NSrNFZ4M

- Being used to build Internet-scale applications
  - Docker, Kubernetes
  - High-performance web APIs to support modern frameworks
    - Django, Rails, NodeJS hard to scale
  - Companies switching over…Google, Microsoft

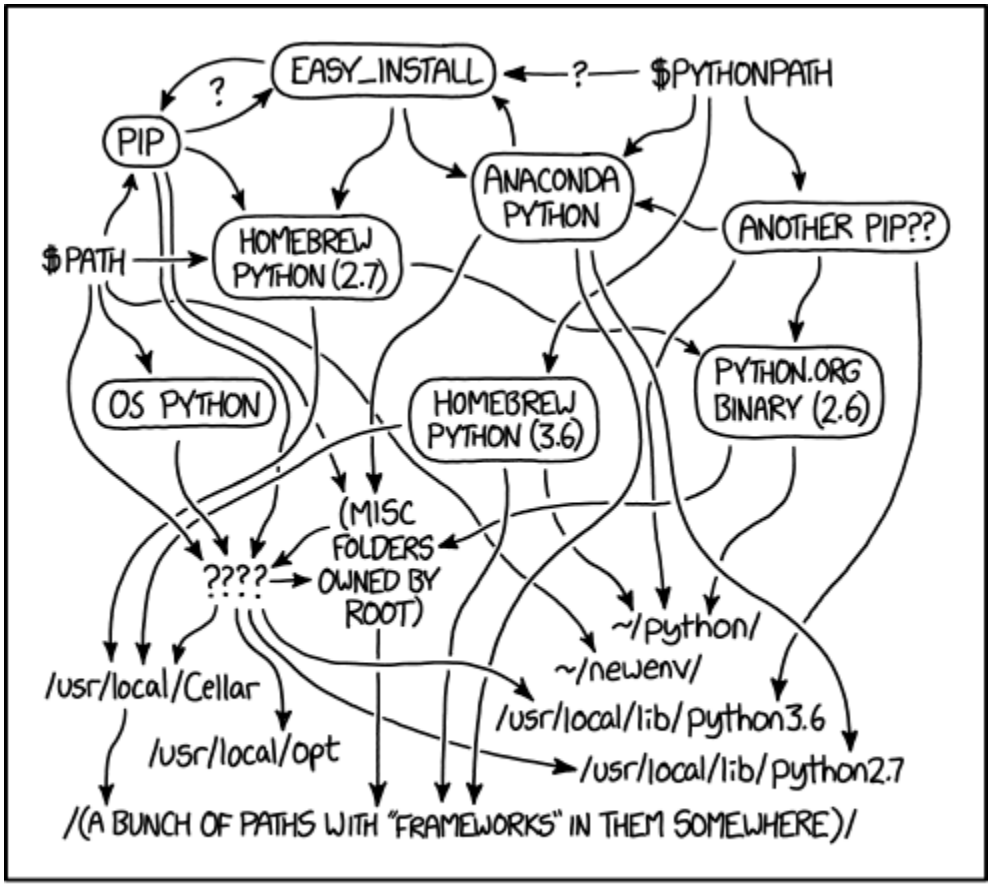
Python

- Interpreted, can be compiled to bytecode (.pyc)
- Dynamically typed (type errors caught at run-time)
- Strongly typed (type conversion must be explicit)
- Managed memory (garbage collected)
- async support in Python 3.6
- Not built with concurrency in mind
  - Global interpreter lock
  - Prevents Python programs from running on multiple cores

- Extensive packages, conciseness, and huge developer base
- Gets you this https://xkcd.com/353/



- But, also this https://xkcd.com/1987/



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

- Deployment requires tools for package (pip) and virtual environment management

Web development (at a 10k level)

- Typical web application components
  - Programming language
  - Framework
  - Template system

- Rendering approaches

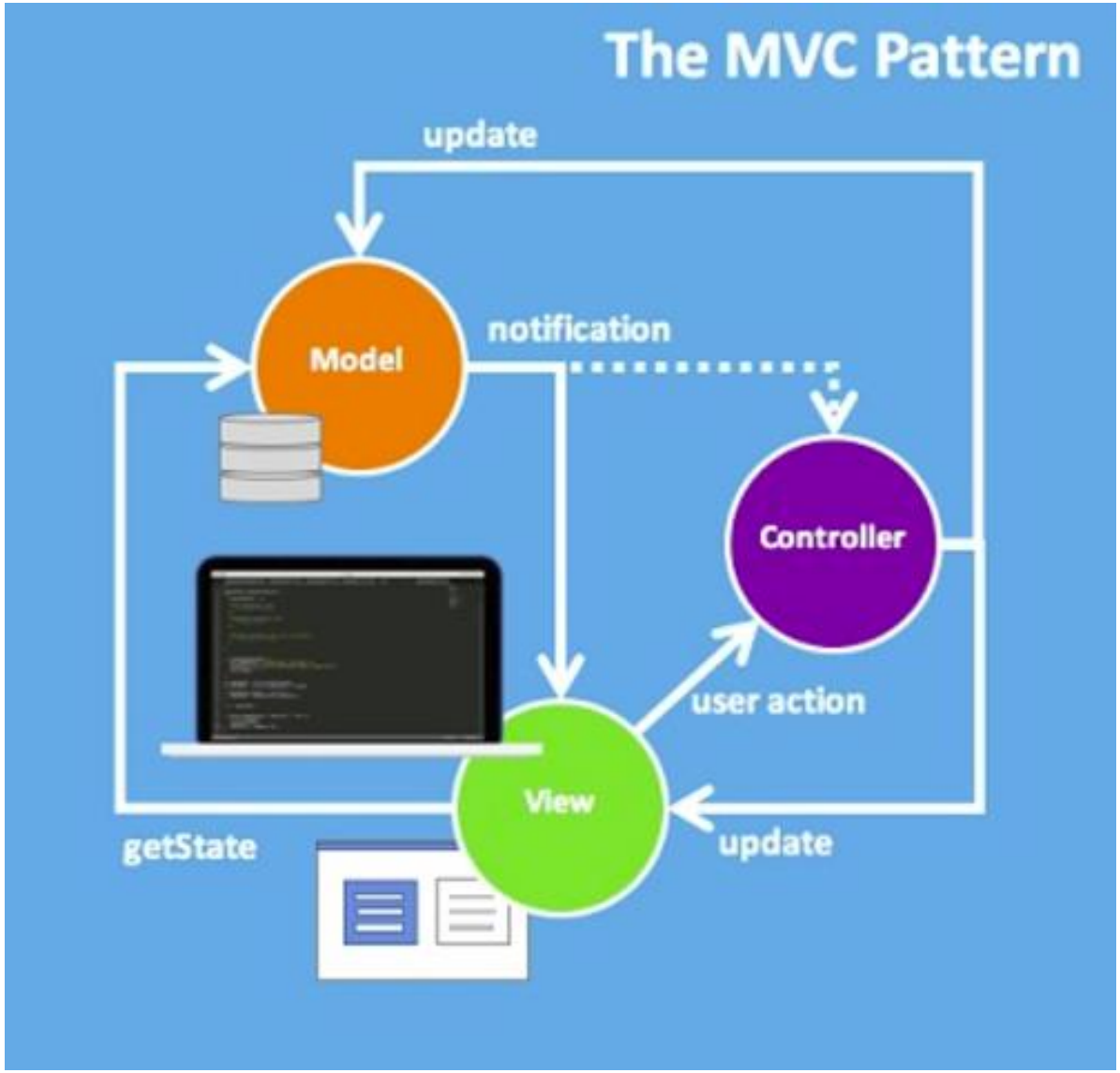
Web frameworks

Web frameworks

- Library support for building complex web applications
- Tied to a programming language
- Supports
  - Basic routing of URLs to code
  - Often implements an "opinion" on how web application should be structured

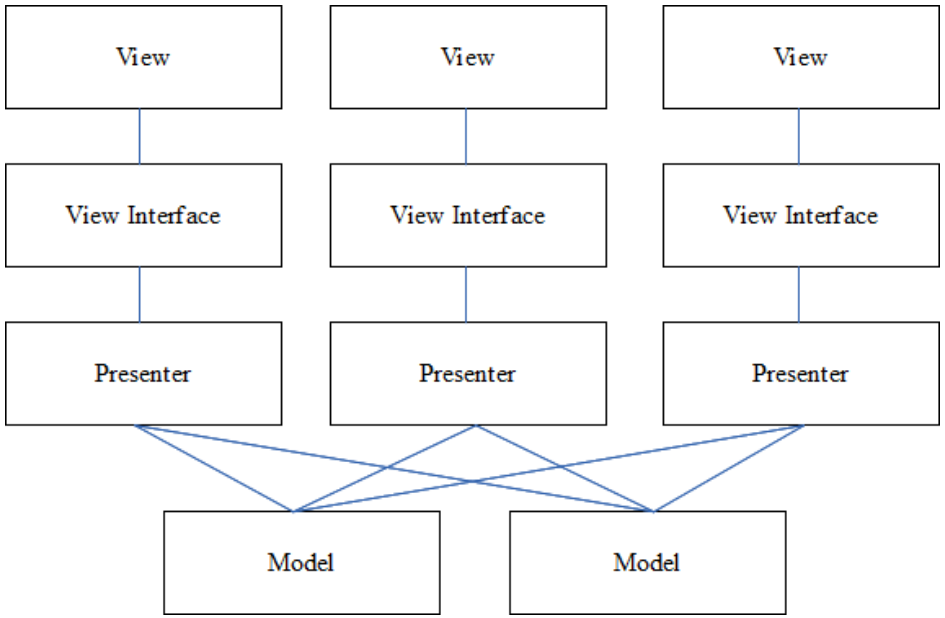
MVC (Model-View-Controller) architecture

- Developed in 1970s (Smalltalk), adapted everywhere
- First used in web applications in late 1990s via Struts
- Model
  - Code that encapsulates backend application data
  - Data representation and storage
- View
  - Code for rendering that generates HTML output and presents functionality to user (UI)
- Controller
  - Code connecting model and view
    - Takes in user requests to update model
    - Pulls in model data to supply the view
- Separation of concerns
  - Simplifies swapping out database backend or the UI frontend
  - Examples of each in repository (cs410c-src)


The MVC Pattern
update
Model
notification
Controller
user action
View
getState
update

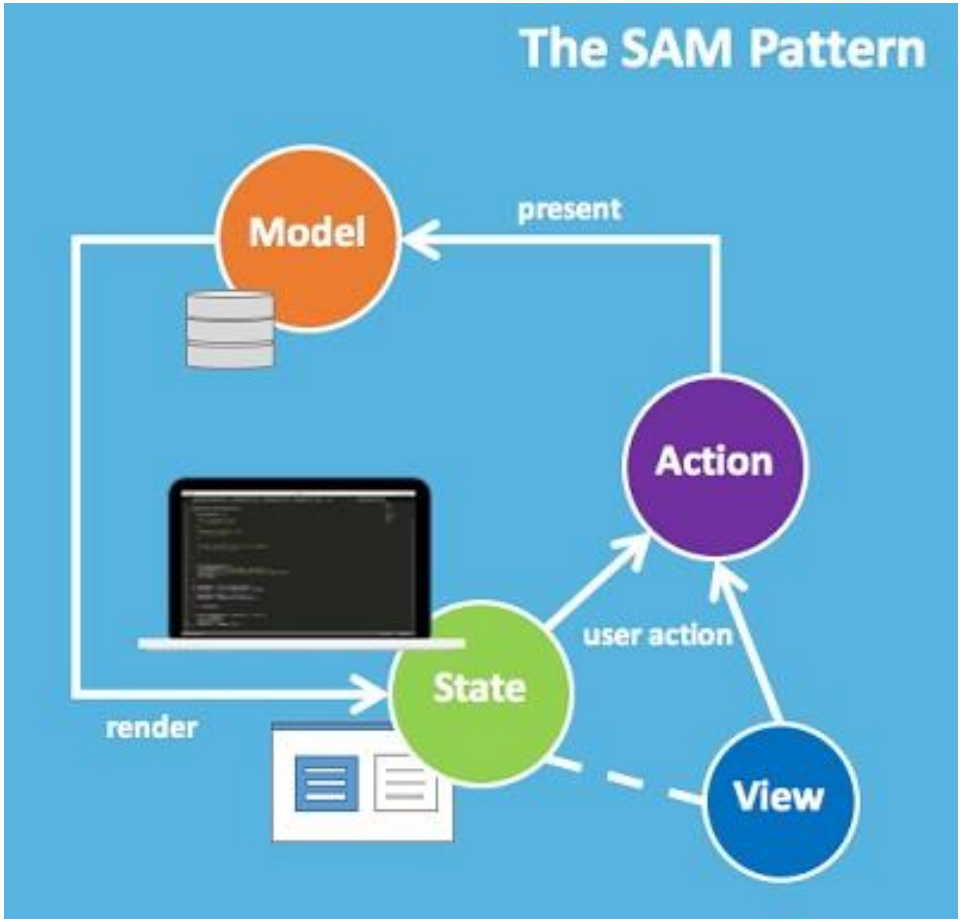
Model-View-Presenter (MVP)

- MVC
  - Controller determines which views to pass back based on user actions
- MVP
  - User actions directly bind to specific "presenters"
  - Presenter handles specific interaction between a view and the model
  - "Function"-based decomposition better suited for unit testing



State Action Model (SAM)

- SAM pattern also removes controller (similar to what React does)
  - https://www.infoq.com/articles/no-more-mvc-frameworks



Web development (at a 10k level)

- Typical web application components
  - Programming language
  - Framework
  - Template system

- Rendering approaches

Template system

Template engines

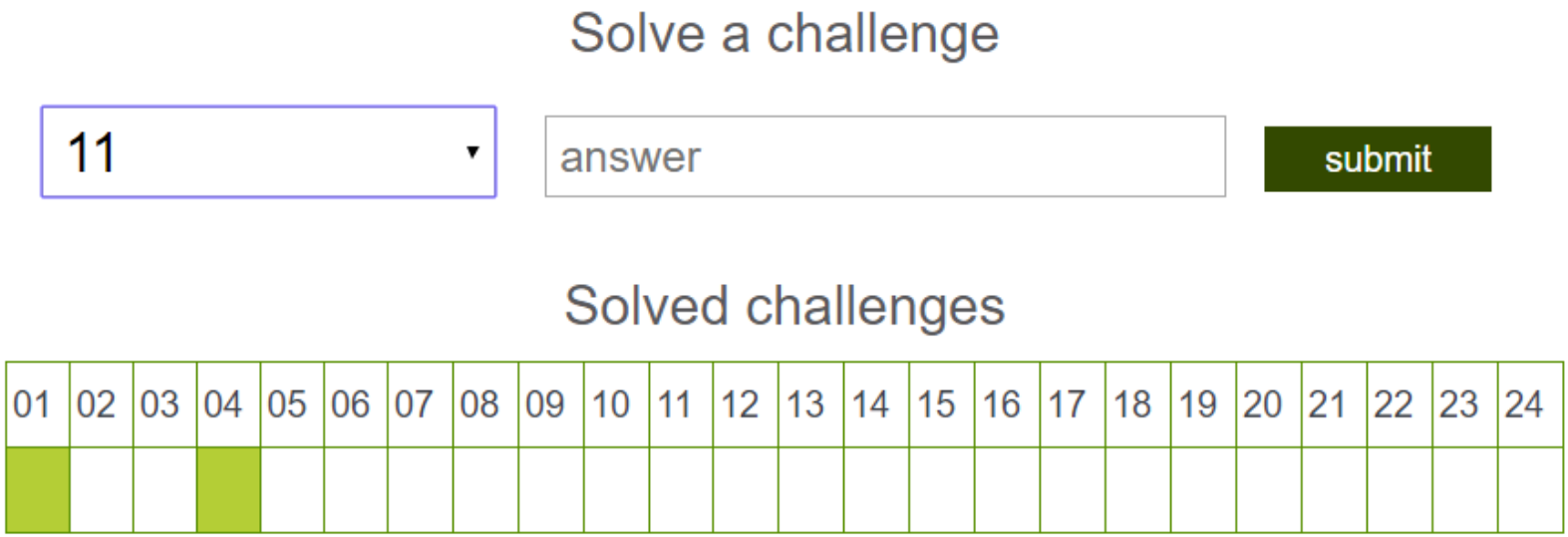
- Web app uses one language (e.g. JavaScript, Python, Java, Go) to produce another language (e.g. HTML)
- Initially, programming language generates entire HTML string
  - Java servlets
- Then, tree-building libraries to construct DOM used to produce the HTML string
  - Repetitive, manual construction of pages
- Then, templating
  - Template language for specifying base page that is filled in dynamically by web application
  - Template engine generates eventual HTML
  - Examples: JSP (Java Server Pages), Jinja2, Mustache
- Separates presentation (view) from the logic (controller)

Example: Jinja2 syntax

- *Extensible* HTML
  - `{{ arg }}` corresponds to template arguments in HTML file
    - Pass `arg=val` to our template to render HTML with data being passed
  - `{% %}` encompasses control statements (`if`, `for`, `block`, etc)
    - Can use data being passed to drive conditionals in templating system
    - e.g.
```
{% if arg %}
    <h1>arg included</h1>
{% endif %}
```

Jinja2 example

- CTF interface for CyberPDX crypto site
  https://crypto.cyberpdx.org
- "Solve" page consisting of
  - Form containing challenges yet to be solved in a drop-down
  - Scoreboard containing shaded challenges already solved



```
class Solve(flask.views.MethodView):
    @login_required
    def get(self):
        sols = [f for f in solved.keys()]
        nsol = [f for f in notsolved.keys()]
        return flask.render_template('solve.html',
                                     solved=sorted(sols),
                                     notsolved=sorted(nsol),
                                     challenges=sorted(challenges.keys()))
```

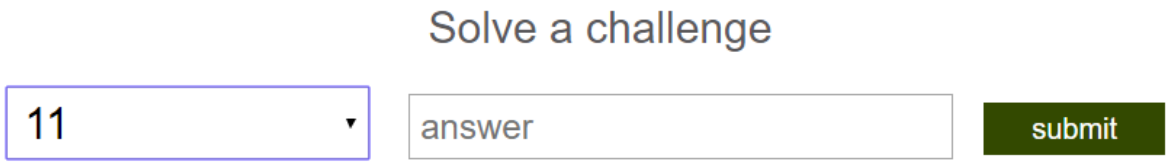
Controller logic for generating lists "solved", "notsolved", "challenges"

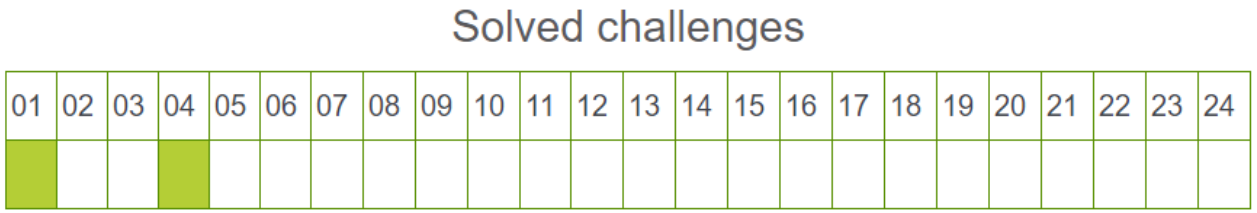
Passes to view via render_template

```
<h2>Solve a challenge</h2>
<form action="{{url_for('solve')}}" method="post">
  <select name="challenge">
    {% for n in notsolved %}
    <option value="{{n}}" />{{n}}
    {% endfor %}
    </option>
  </select>
  <input type="text" placeholder="answer" name="guess" maxlength="20" />
  <input type="submit" value="submit" />
</form>
```

View implemented in templating engine for drop-down form submission using `notsolved`

`solve.html` Jinja2 template





`solve.html` Jinja2 template

..scoreboard table in template using `challenges` & `solved`

```
<h2>Solved challenges</h2>
<table class="score">
  <tr>
    {% for c in challenges %}
    <td width=3.5%>{{c}}</td>
    {% endfor %}
  </tr>
  <tr>
    {% for c in challenges %}
       {% if c in solved %}
            <td class="green">   </td>
       {% else %}
            <td>   </td>
       {% endif %}
    {% endfor %}
  </tr>
</table>
```

```
<h2>Solve a challenge</h2>
<form action="{{url_for('solve')}}" method="post">
  <select name="challenge">
    {% for n in notsolved %}
    <option value="{{n}}" />{{n}}
    {% endfor %}
    </option>
  </select>
  <input type="text" placeholder="answer" name="guess" maxlength="20" />
  <input type="submit" value="submit" />
</form>
```

Form does a POST when submitted with data named "`challenge`" and "`guess`"
Controller logic for processing solves pulls out "`challenge`" and "`guess`" in form

```
@login_required
def post(self):
    ...
    username = flask.session['username']
    challenge = flask.request.form['challenge']
    guess = flask.request.form['guess']
    ...
    answer=challenges[challenge]
    if answer==guess:
        flask.flash("Correct.  You have solved " + challenge)
        ...
```

Templating issues

- Now 3 programming languages to learn and debug together!
  - HTML, web app, template
  - Program logic split between template language and web application language
  - Why was the "green" condition parsed in Jinja2 and not Python?
  - Where should the conditional rendering of content go?
    - All in the template (e.g. view)?
    - All in the controller?

Templating via DSLs

- Internal domain-specific languages
  - Templating language integrated with the host language
- Examples
  - JSX (native XML support for JavaScript) (React)
  - Kotlin (JetBrains)
    - Statically typed (i.e. type-safe) HTML builder for JavaScript or Java
  - Elm, Hyperscript, Groovy, Flutter, etc.
- But, need to be careful
  - Want separation of concerns
  - Control logic must be separated from UI logic within language

https://medium.com/@daveford/80-of-my-coding-is-doing-this-or-why-templates-are-dead-b640fc149e22

Summary survey

Language	Framework	Template
Java	Java servlets (minimal) Spring, Apache Struts (MVC)	Java server pages
Javascript/NodeJS	Express (minimal) Sails, AngularJS (MVC)	Mustache, Handlebars
Python	Flask (minimal) Django (MVT)	Jinja2, Django templates
C#	ASP.NET routing (minimal) ASP.NET MVC	Razor
Ruby	Sinatra (minimal) Rails (MVC)	Slim, HAML
Go	Echo (minimal) Martini, Beego (MVC)	Amber, Mustache
PHP	Fat-free, Slim (minimal) Laravel, Symfony (MVC)	Blade, Mustache

Web development (at a 10k level)

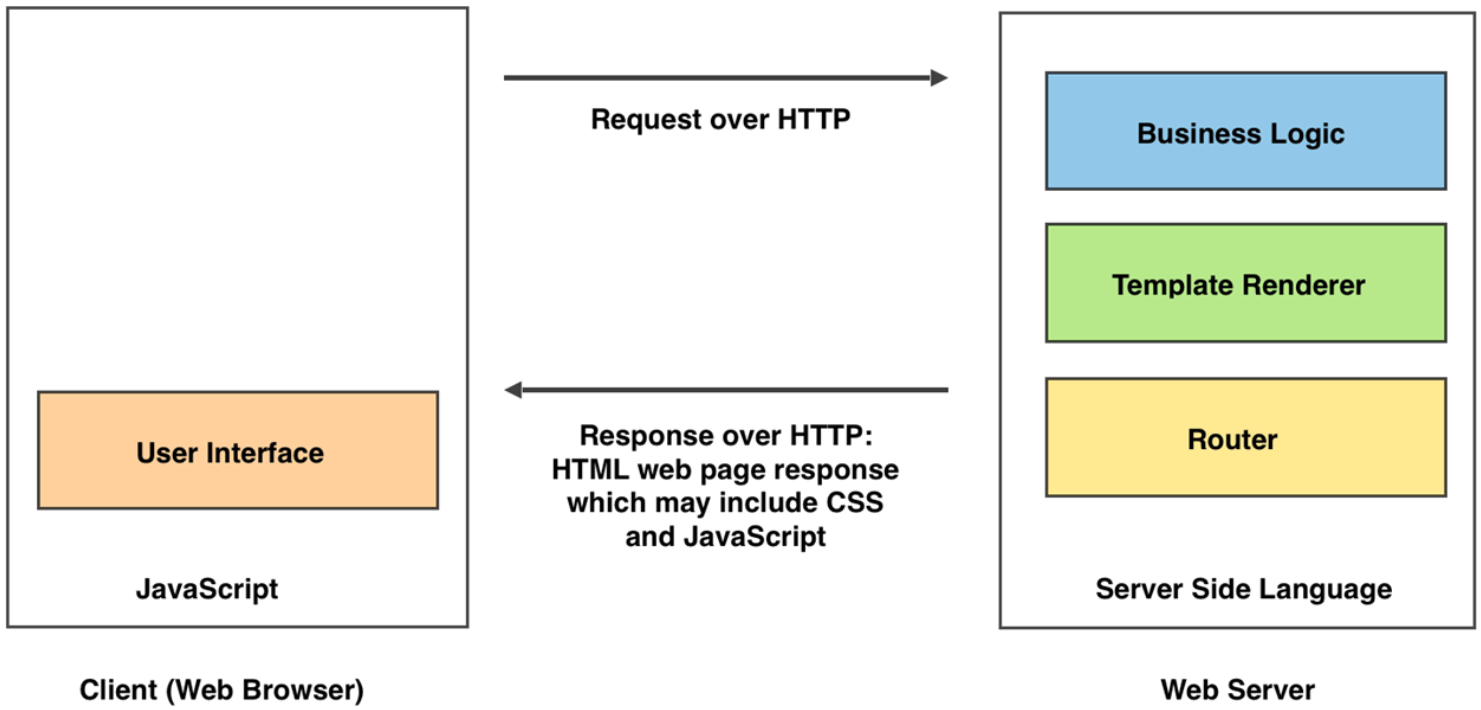
- Typical web application components
  - Programming language
  - Framework
  - Template system

- Rendering approaches

Rendering approaches

Rendering

- Two rendering steps
  - Web application renders server data into HTML/CSS for web browser
  - Web browser renders HTML/CSS for user
- Initially, MVC and MVP all running on the server
  - Your app for this class
  - (Figures used from K. Balasubramanian, "Isomorphic Go")



Issue #1: Repetitive server rendering

- Dynamic-ish content
  - What if articles on a web application only change once a day?
  - Example: WordPress blog updated daily
    - Executes web application logic and hits backend database to generate HTML, even though results are the same all day long!
- Server does much more work than necessary

Pre-rendered pages

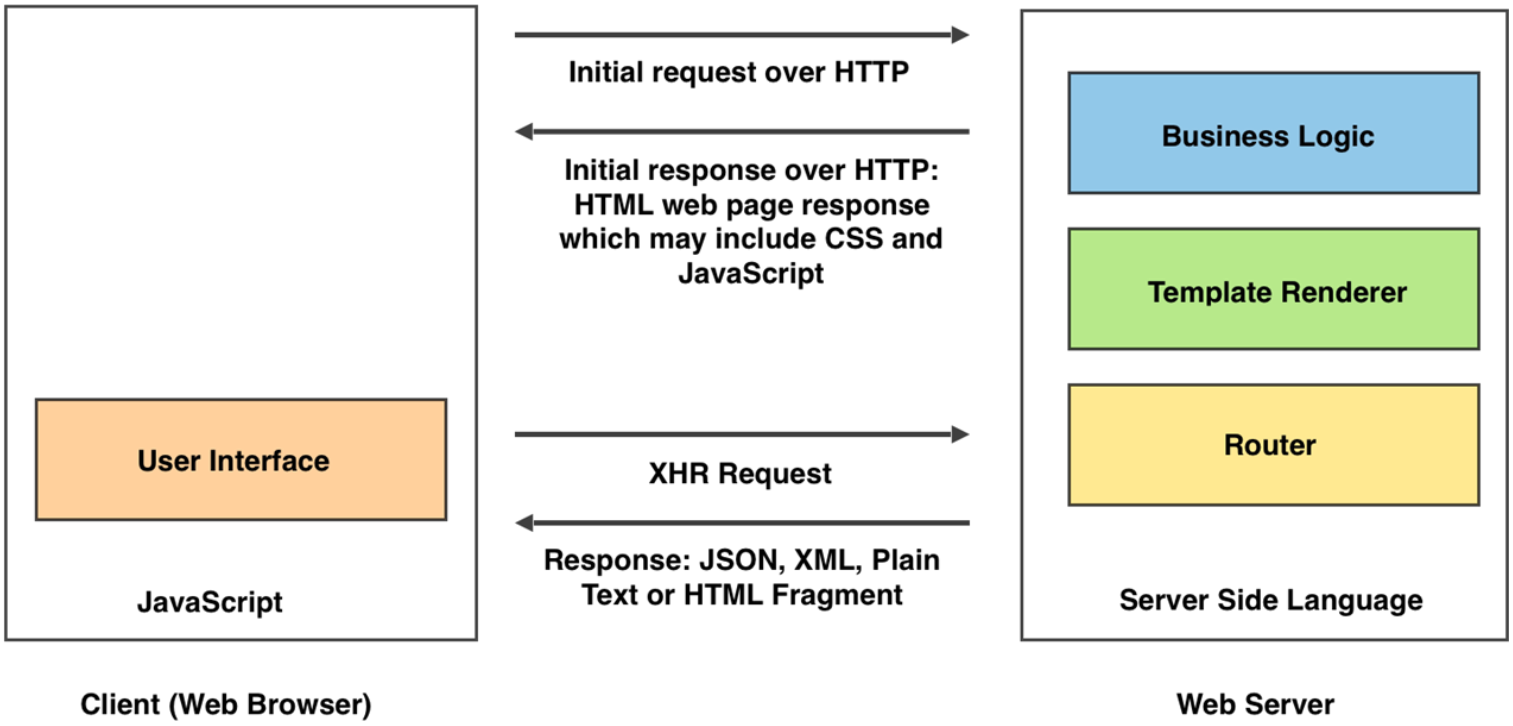
- General approach of handling "dynamic" content that does not change
  - Server pre-renders dynamic page into static HTML
  - Static page can be forward deployed (via CDN) and cached for performance
  - Can be returned to search engines and easily crawled for indexing
  - More secure! (Clients interact with static content)
- Supported now in most frameworks

Issue #2: Interactivity

- User navigation hits web application running on the server each time
  - Poor interactivity, page reload on every action
- Motivates a push towards client-side rendering and client-side control of web application rendering

Client-side rendering

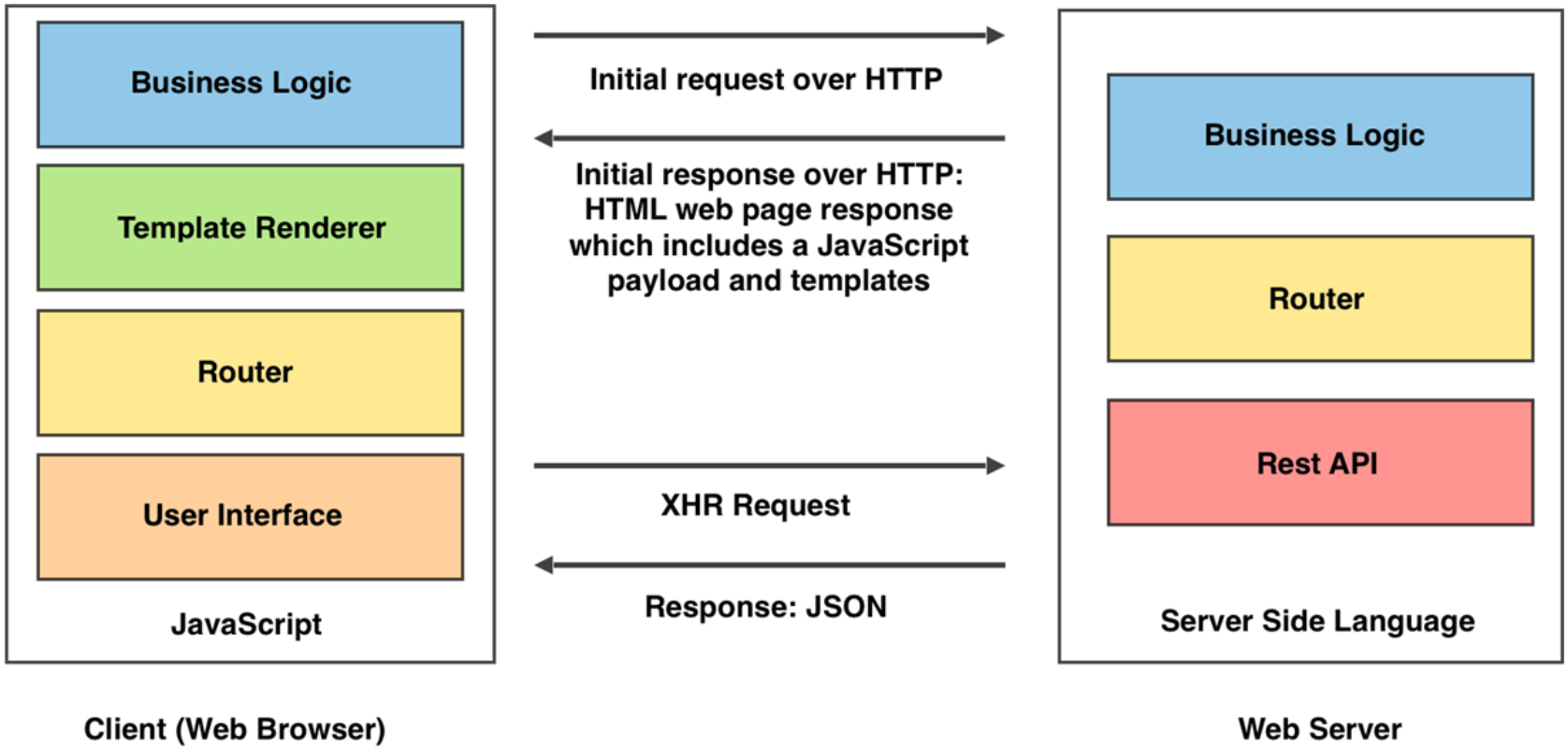
- Allow client to retrieve and update HTML/DOM elements
- First instance: AJAX (Asyncrhonous JavaScript and XML)
  - JavaScript library code for reducing server-side rendering of content
    - Update DOM in place via HTTP request from JavaScript running on page



- Eventually leads to more voperations placed into browser
  - jQuery, Bootstrap, Meteor, etc.
- Benefits
  - Client-side operations more responsive (reduce page reloads)
  - Can have just the changed portions of page downloaded
  - Fewer connections and decreased load on server
- Can one render and ship the entire application and push it to the client?

Single-page applications

- Ship entire web application to client in a single package
  - Controller, View, and page rendering all on client
  - Typically, a single `index.html` file, a CSS bundle, and a JavaScript bundle
  - Examples: GMail, Google Maps, Facebook, Github
  - Backend only supplies model via an API
    - e.g via REST/gRPC or GraphQL (React)
  - Web app view directly interacts with API
    - Controller removed
      - Angular (Google) "MVW (model-view-whatever)"
      - Vue "MVVM (model-viewmodel-view) similar to MVP"
- Advantages
  - Essential static content delivered ahead of time with no need to refresh (helped by HTTP/2 server push)
  - Faster UI since network calls minimized
- Disadvantages
  - Initial load time for heavy client frameworks

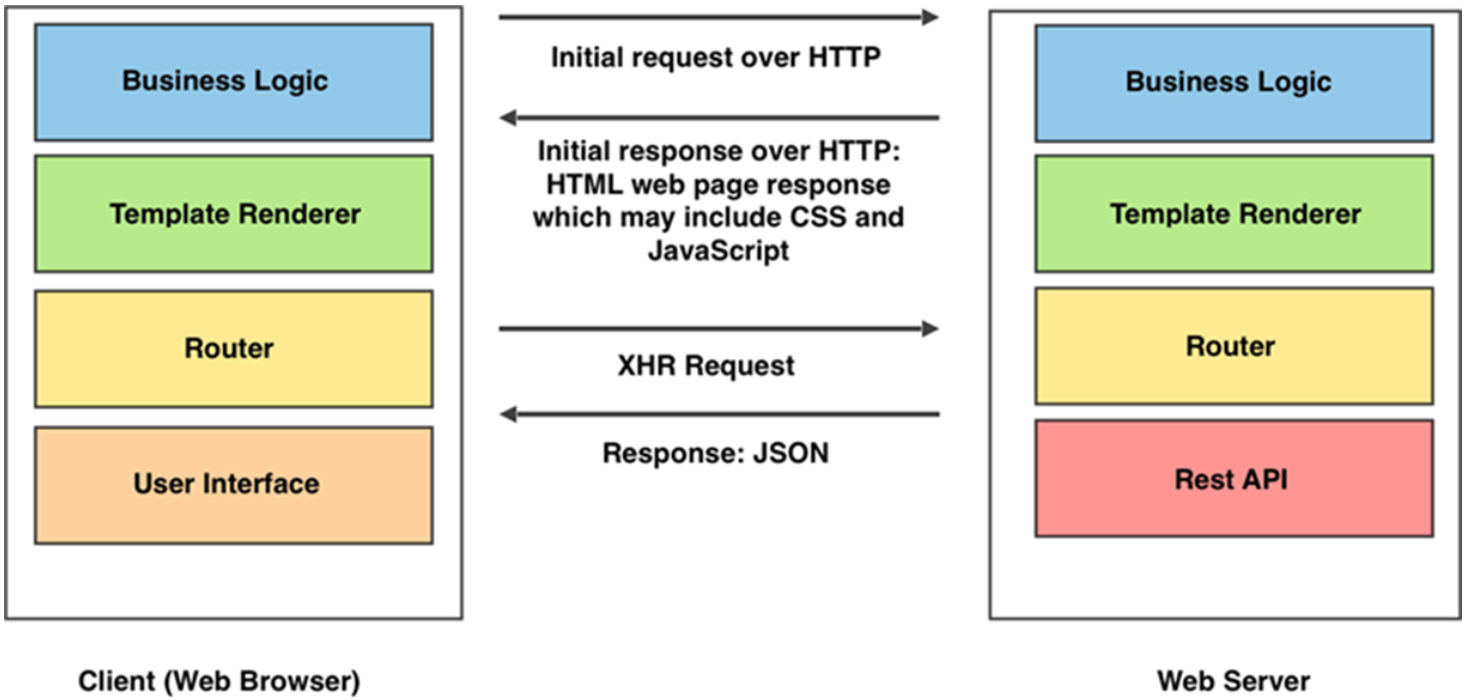

Business Logic
Template Renderer
Router
User Interface
JavaScript
Client (Web Browser)
Initial request over HTTP
Initial response over HTTP: HTML web page response which includes a JavaScript payload and templates
XHR Request
Response: JSON
Business Logic
Router
Rest API
Server Side Language
Web Server

- Drawbacks
  - Not good for search-engine optimization (SEO)
    - Web scapers typically do not run JS engine
  - Complexity at the client
    - Package management, bundling, minification, component libraries, cache busting, bundle splitting, automated testing/building
  - CS 465P/565
- Can we render a version for search-engines to index that is identical to what a client uses?

Isomorphic web pages

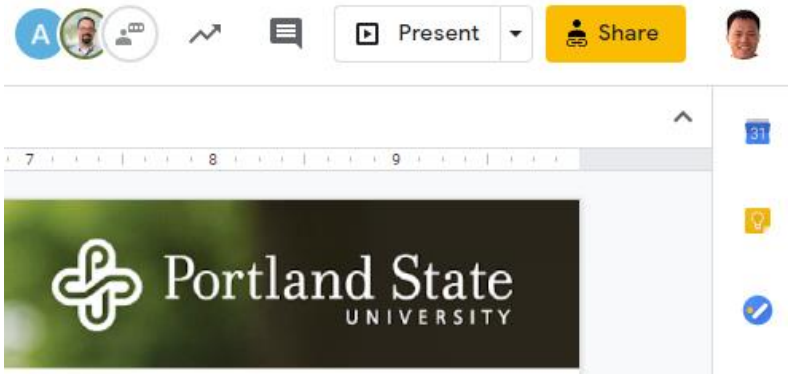
- Latency an issue with single-page applications
- Hybrid approach
  - Serve initial page via traditional server rendering or via pre-rendered version
  - Ship full client-side SPA in the background that supports isomorphic pages
    - Pages that render the same at either client or server
  - Seamlessly switch over once SPA downloaded
  - Benefits
    - Fast initial load
    - Good with SEO
    - Eventual responsiveness that SPA provides
  - Example: Angular Universal
    - https://blog.angular-university.io/why-a-single-page-application-what-are-the-benefits-what-is-a-spa/
    - https://medium.com/airbnb-engineering/isomorphic-javascript-the-future-of-web-apps-10882b7a2ebc

- Rendering on both sides



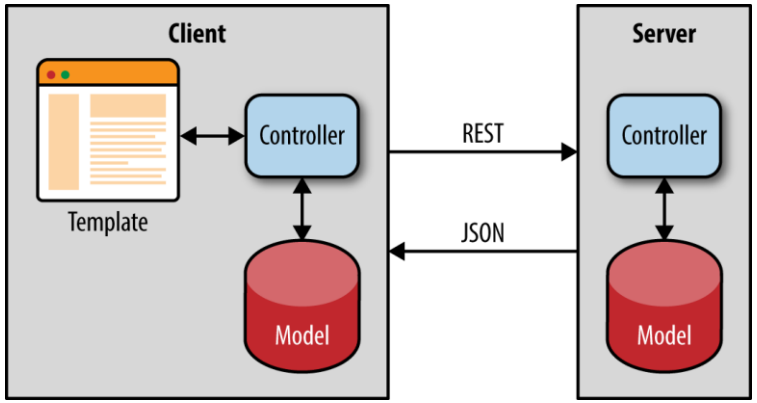
Isomorphic web applications

- Collaborative multi-user applications
  - State shared amongst servers and multiple clients
  - Examples:

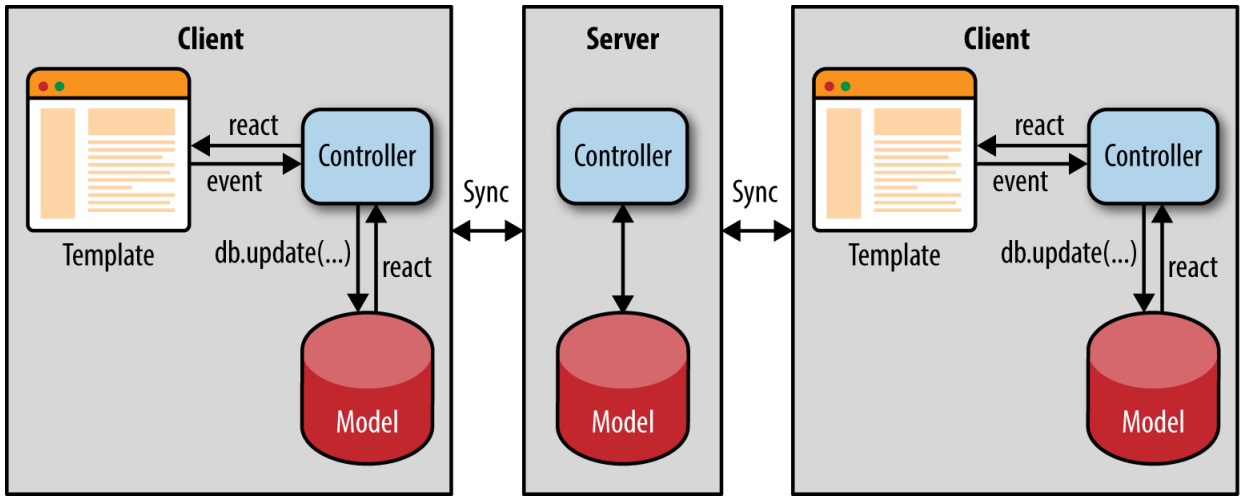


  - Want a shared real-time database with isomorphic rendering done with minimal overhead

- Limits to updates and rendering via REST/JSON
  - https://www.oreilly.com/library/view/building-isomorphic-javascript



- Real-time clients with bidirectional synchronization
  - Synchronize models to each other via alternate means (e.g. WebSocket)
  - Meteor.js "Database Everywhere"

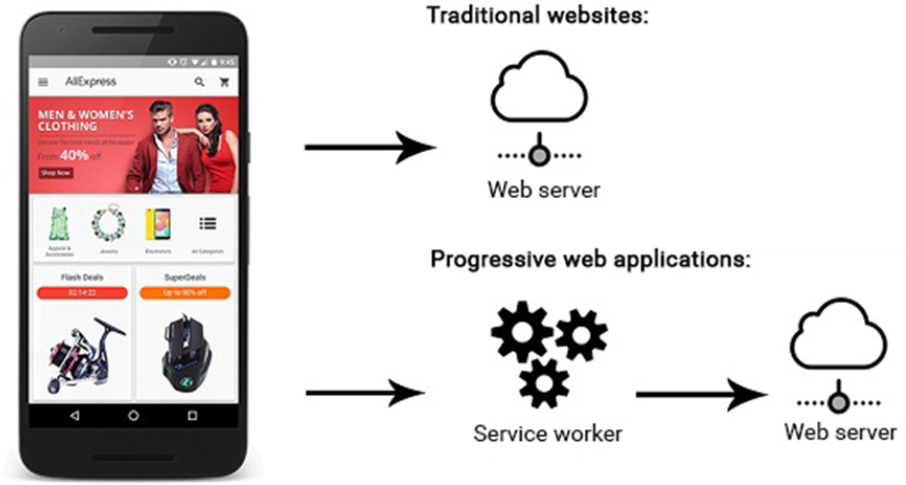


Issue #3: Disconnected operation

- Web app done as an SPA close to a native application
- But, want to provide a Microsoft Word experience to something like a Google Doc
  - Want web app (Google Doc) to seamlessly support disconnected operation and act more like a native desktop (or mobile) app

Progressive web applications (PWA)

- Web applications doubling as native desktop/mobile applications
  - Seamlessly handle off-line, disconnected operation
  - Sync state upon reconnection



- Key abstraction: Service workers
  - Sit between web applications and the browser
  - Intercept network requests and take appropriate action based on whether the network is available
  - Receive notification when updated assets reside on the server
- Makes heavy use of JavaScript Promises
  - Promise represents the eventual completion (or failure) of an asynchronous operation, and its resulting value

Issue #4: Too many frameworks

- Developers re-inventing wheels
  - Modules in React, Angular, Vue, … provide similar functions but are incompatible
- Goal: Implement a component model for modularity and reusability
  - Turn web application a collection of type-checked re-usable components
  - Standardized via Web Components in W3C
    - Wrap client UI widgets in a standard way
    - Example: specify a COTS component that gets an input, and after some internal behavior / computing, returns a rendered UI template (a sign in / sign out area or a to-do list item) as output
    - https://codewithhugo.com/how-components-won-the-framework-wars/

Preparing for Homework #2