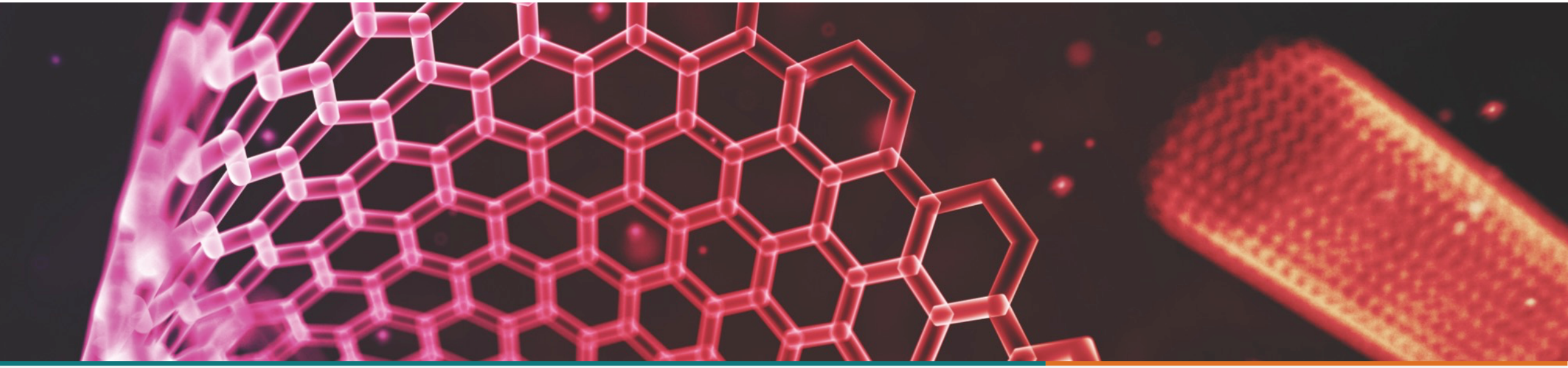
STEVENS
INSTITUTE of TECHNOLOGY
Schaefer School of Engineering & Science

CS 554 – Web Programming II

ES6 and Fundamentals of Web Development



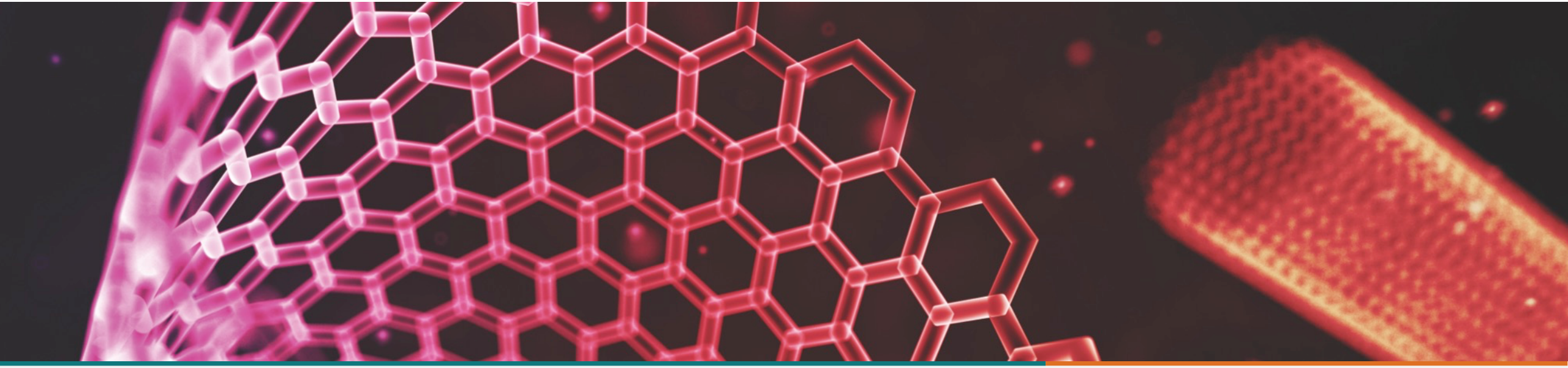
STEVENS
INSTITUTE of TECHNOLOGY

Schaefer School of
Engineering & Science

stevens.edu

Patrick Hill
Adjunct Professor
Computer Science Department
Patrick.Hill@stevens.edu

Course Technologies



Our Tech Stack

By default, we will be starting with the Node, Express, Mongo tech stack

- Direct continuation of Web 1 web stack, with tons more!

Most instructions will assume a Mac OS X operating system in this course.

Recommended Operating System

Windows is **not** a recommended operating system for this course, and Mac OSX will be the assumed OS in all instructions due to it being the ideal web development environment. Linux is also great for web development

If you do not have a Mac system, you can emulate it using VirtualBox

- https://techsviewer.com/install-macos-sierra-vmware-windows/

If you prefer Linux:

- https://theholmesoffice.com/installing-ubuntu-in-vmware-player-on-windows/

Node Version

In this course, we will be using Node for our server side language. You should install the most recent current version of Node for this course.

- https://nodejs.org/en/download/current/

You may find it easier to install with NVM to maintain many versions of node

- https://github.com/creationix/nvm
- https://github.com/coreybutler/nvm-windows

MongoDB

For most scenarios, we will be using MongoDB for the de facto database of choice.

- https://www.mongodb.com/download-center/community

It is an easy, simple, key-value database.

Other Course Technologies

Redis:

- https://redis.io/

Gulp:

- https://gulpjs.com

React:

- https://reactjs.org

GraphQL:

- https://graphql.org

Next.js:

- https://nextjs.org

Vue.js:

- https://vuejs.org

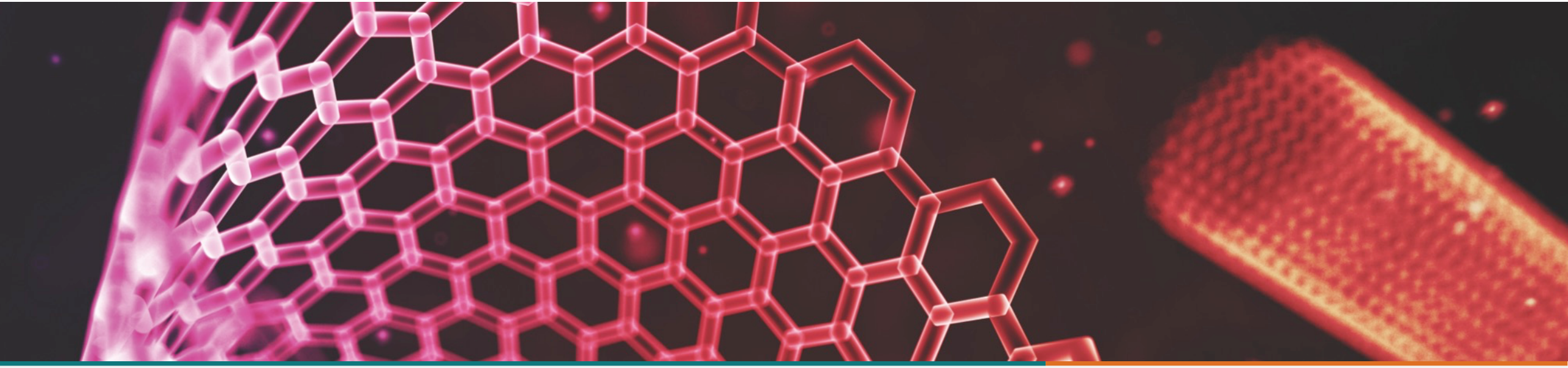
TypeScript:

- https://www.typescriptlang.org

Socket.io:

- https://socket.io

ES6



What is ES6?

ES6 is the most recent spec of what constitutes the JavaScript language. It is, at this point, mostly adopted on the Node side.

ES6 provides a series of very large conveniences, especially with easier asynchronous code handling.

Asynchronous Code Is Messy

In general, writing asynchronous code is syntactically messy.

- Ends up in deeply nested callbacks
- Creates huge promise chains
- Hard to error check

As a result, over the last several years, the JavaScript language has added a concept of **async functions and await**ing the end of an asynchronous operation.

Async Functions

A function can be defined as an `async function`, which means that the function will automatically return a promise (even if there are no asynchronous operation in that function).

- By marking a function as `async`, you allow the `await` keyword to be used inside of the function.

Besides their ability to `await` promises and their guaranteed returning of a promise, there is no difference between a *function* and an *async function*.

You can ONLY use the `await` keyword inside an `async` function.

Awaiting Promises

The `await` keyword can only be used inside of an `async function`

When you `await` a promise, you will cause the rest of the function to execute after that promise resolves or rejects. If the promise rejects, an error will be thrown on the line that awaits it.

- This allows you to use `try`/`catch` syntax in asynchronous operations!
- The result of an `await` operation is whatever the promise resolves to. If the promise does not resolve to a value, it will have a result of `undefined`.

Example

On the top, we see promises. On the bottom, we see the same code written in ***async***/***await*** notation.

```
12  prompt
13    .getAsync([getFileOperation])
14    .then(function(result) {
15      const fileName = result.fileName;
16      if (!fileName) {
17        throw "Need to provide a file name";
18      }
19
20      console.log(`About to read ${fileName} if it exists`);
21
22      return fileName;
23    })
24    .then(function(fileName) {
25      return fs.readFileAsync(fileName, "utf-8").then(function(data) {
26        return { fileName: fileName, fileContent: data };
27      });
28    })
```

```
8   async function main() {
9     const getFileOperation = {…
12    };
13
14    // Gets result of user input
15    const promptResult = await prompt.getAsync([getFileOperation]);
16    const fileName = promptResult.fileName;
17
18    if (!fileName) {
19      throw "Need to provide a file name";
20    }
21
22    console.log(`About to read ${fileName} if it exists`);
23    const fileContent = await fs.readFileAsync(fileName, "utf-8");
24
```

What Benefit Does It Have?

By using `async` and `await` we can create code that is written and read in the order that which the operations will complete, while still allowing the functions themselves to be asynchronous.

Internally, the functions that run async code will still be constantly giving up execution cycles to perform other operations while they continue their tasks, however your code can abstract over that.

Default Input Parameters

Now, we can set default values to our parameters. If a parameter is undefined, it will take on the default value.

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Default_parameters

```
function addTwoNumbers(num1 = 1, num2 = 2) {
  return num1 + num2;
}

console.log(addTwoNumbers()); //Prints out 3
console.log(addTwoNumbers(3)); //Prints out 5
```

Template Literals/String Interpolation

Rather than concatenating strings like barbarians, we can now just interpolate our strings.

We can also make multiline strings in the same format. Finally.

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals

```
let hello = 'Hello';
let world = 'World';
//old
console.log(hello + ' ' + world);
console.log(hello + '\n\n\n' + world);
//new
console.log(`${hello} ${world}`);
console.log(`${hello}


${world}`);
```

Output:

```
Hello World
Hello


World
Hello World
Hello


World
```

Object Destructuring

Say we have the following object that has a nested array of objects and an array within it:

```
let myobj = {
  name: 'Patrick Hill',
  education: [
    { Level: 'High School', Name: 'Bayside High School' },
    { Level: 'Undergraduate', Name: 'Baruch College' },
    { Level: 'Graduate', Name: 'Stevens Institute of Technology' }
  ],
  hobbies: [ 'Playing music', 'Family' ]
};
```

If we want to store the array of objects in education into a variable, and the hobbies into another variable. We can use the following syntax:

```
let {education, hobbies} = myobj
```

Then we can access JUST the education or hobbies from the object using the variable:

```
console.log(education);//Will print out the education from the object

console.log(hobbies);//Will print out the hobbies from the object
```

```
[
  { Level: 'High School', Name: 'Bayside High School' },
  { Level: 'Undergraduate', Name: 'Baruch College' },
  { Level: 'Graduate', Name: 'Stevens Institute of Technology' }
]
[ 'Playing music', 'Family' ]
```

More info on object destructuring: https://wesbos.com/destructuring-objects/

Destructuring Arrays

Similarly, arrays work the same way.

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment

```
1 | var foo = ['one', 'two', 'three'];
2 |
3 | var [one, two, three] = foo;
4 | console.log(one); // "one"
5 | console.log(two); // "two"
6 | console.log(three); // "three"
```

Object Spread

While destructuring, we can also spread an object and create another object with those keys.

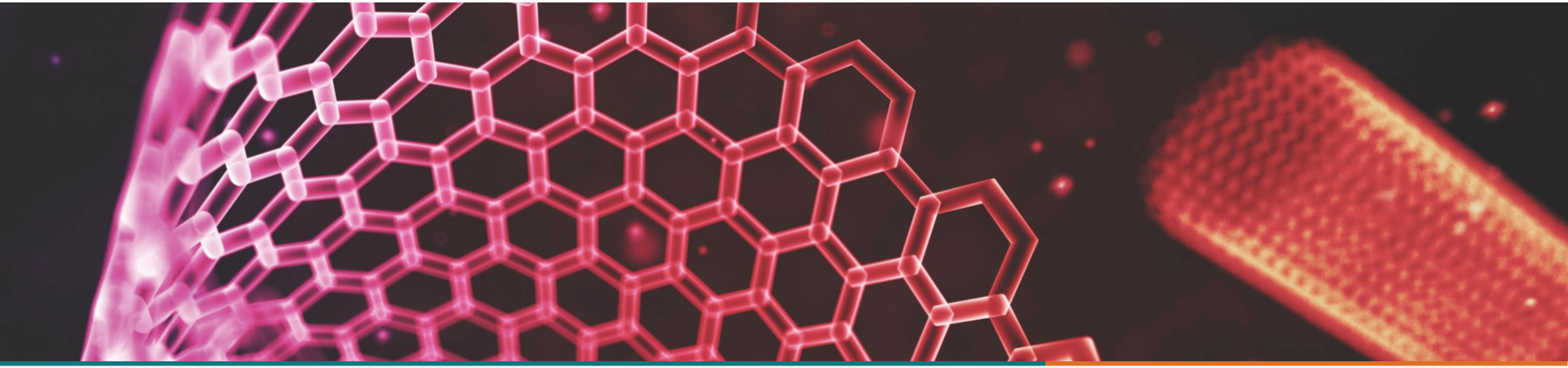
This also makes for good shallow cloning, as well. It is not a deep clone operation. You can shallow-clone an object by using the spread operator without other parameters, or you can override

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_operator

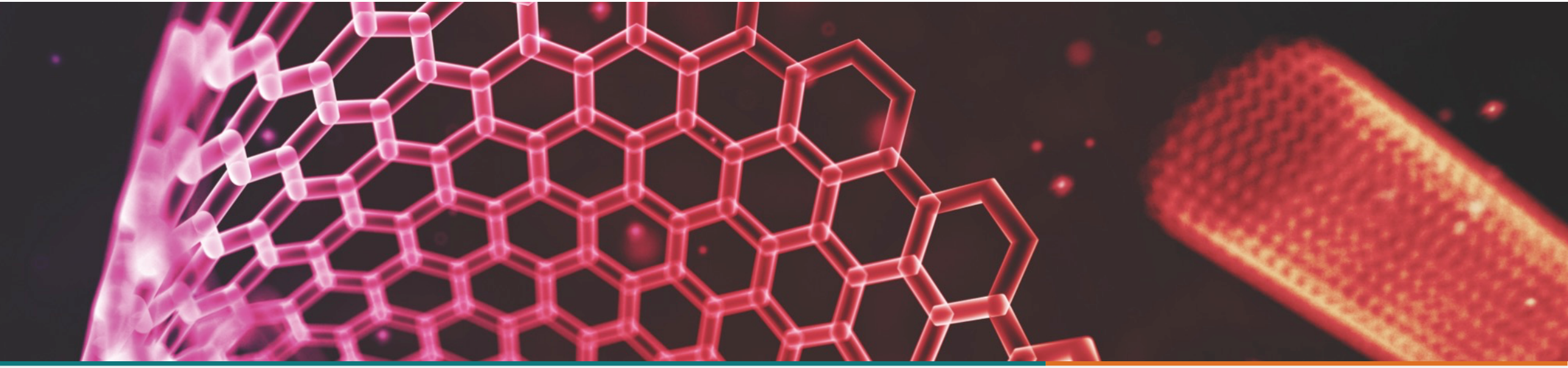
```
let {a, b, ...rest} = {a: 10, b: 20, c: 30, d: 40}
a; // 10
b; // 20
rest; // { c: 30, d: 40 }
```

```
var obj1 = { foo: 'bar', x: 42 };
var obj2 = { foo: 'baz', y: 13 };

var clonedObj = { ...obj1 };
// Object { foo: "bar", x: 42 }

var mergedObj = { ...obj1, ...obj2 };
// Object { foo: "baz", x: 42, y: 13 }
```

Web Development Recap



Intro to JavaScript and Node



Some Basic Facts About JavaScript

JavaScript is a loosely typed language, a concept that you may have seen before.

- Loose typing means that you don't strictly declare types for variables, and you can change the type of data that you store in each variable.
- Due to this loosely typed nature, you will need to check all inputs for
    1. If there is any input at all.
    2. The input is the correct data type.
    3. The input is in the correct range.

Some Basic Facts About JavaScript

There are five primitives currently, with a sixth (Symbol) on the way

- Boolean
- Number
- String
- Null
- Undefined

JavaScript also has Objects, which all non-primitives fall under

- Functions in JavaScript are types of Objects
- Objects are prototypical

Defining Variables

There are three keywords used to define variables in JavaScript:

Keyword	Scope	Explanation	Example
const	Block	Initializes a block scoped variable that cannot get overwritten. Most common used in this course.	const twelve = 11 + 1;
let	Block	Initializes a block scoped variable that can get overwritten.	let firstName = 'Patrick';
var	Functional	Initializes a variable that can get overwritten; **not used in course.**	var lastName = 'Hill'

Functional Scope In JavaScript

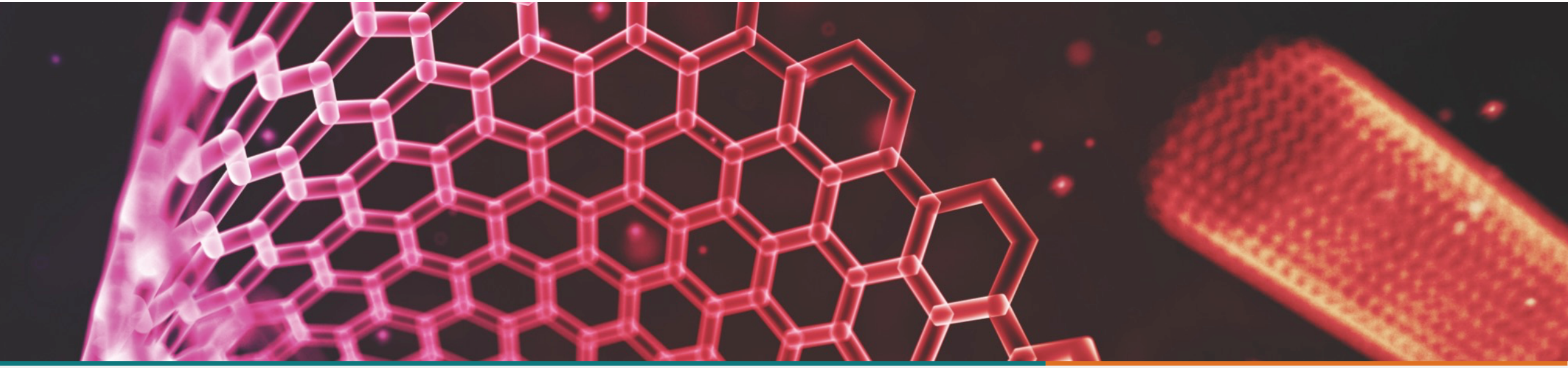
We often want to isolate our scope in JavaScript, particularly when we write browser-based JavaScript code in order to avoid conflicts between libraries.

While Node.js scripts isolate their variables between files, all top-level variables in a browser-environment become global variables, even across different files.

In JavaScript, scope is not defined by block unless using the keyword ***`let`*** to define a variable; when using ***`var`***, it is defined by the function you are in.

- https://hackernoon.com/understanding-javascript-scope-1d4a74adcdf5

MongoDB



What Is a Database?

A database is an organized collection of data. It allows you to create, read, update, and delete data. Unlike storing in memory, databases allow you to persist your data.

MongoDB is a **document-based database**.

- **Document-based** databases store semi-structured data (think, JSON!) and only lookup by key (a unique identifier)

You interact with MongoDB by submitting queries to your database that describe operations that you wish to do.

What Is a Document-Based Database?

Traditional databases are stored in tables that are composed of columns describing data and rows of data. They have a pre-defined schema to them, constraining the type of data you can store in each table.

Document-based databases forgo this and allow you to simply store and retrieve data at a particular location. They are very good at ID lookups but suffer slightly on querying.

This is less off an issue now than in previous years.

Why MongoDB?

MongoDB is incredibly easy to setup and use, allowing you to focus on web-development as a whole and how all the parts work together, rather than focusing on the nuances of databases.

It stores JSON-like structures, making it easy to conceptualize.

- Easy to have nested objects (subdocuments)

Much like objects, each collection stores data in a dictionary fashion using a key/value pair.

The querying language is composed by JSON that describe queries, making it easy to pick up.

The Structure of MongoDB

MongoDB only has a few layers to it:

1. **Databases**: You can create a database to contain related collections
2. **Collections**: Each database has a number of collections. Collections are sets of documents that you decide are related by their content. Your documents do not have to have the same fields.
3. **Documents**: Documents are self contained pieces of data that you store in a collection. Each document must have an ID and can have any other set of fields; these fields can be smaller subdocuments.
4. **Subdocuments**: A document field can describe another document that will be stored in its parent. This is akin to an object that has a second object stored as a property. This is referred to as a subdocument.

Abstracting Your Queries

It is often very useful to create a file to abstract away your database querying.

By creating a layer between your application code and your database, you will allow yourself to:

- More easily make changes to the database later on
- Allow non-database programmers to more easily use the database (separation of concerns!)
- More easily improve performance at a later time
- Easier and more consistent error checking throughout your entire application.
- Make it a reasonable task to change your entire database when the first database your company chose ends up being unable to support large amounts of data and you need to transition over to another database.
  - Also helps when this happens again 2 years later.

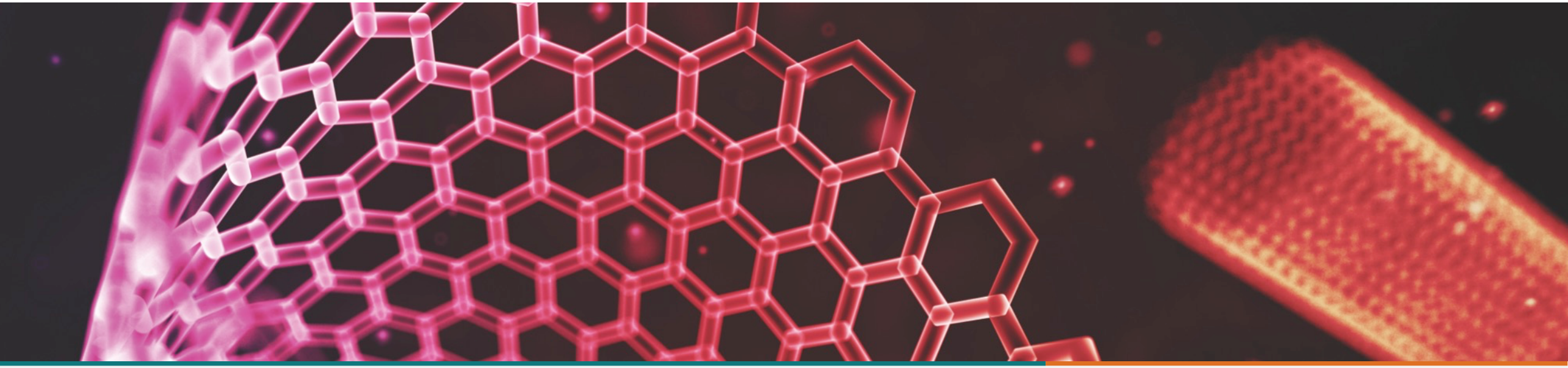
In `dogs.js` we abstract the queries to hide them away from the rest of our application.

Basic Operations in MongoDB

For now, we will be focusing on four different operations:

1. **Insert**: will take an object and insert it into the database.
2. **Find**: will take an object describing fields and values to match and returns an array of matching documents.
3. **Update**: will take two objects; one that contains an object describing fields and values to match, and one that will describe the update to perform. It can update multiple if you provide a third object with settings telling it to update multiple documents.
4. **Remove**: will take an object describing fields and values to match and will remove the object. It can remove multiple if you provide a second object with settings telling it to update multiple documents.

Fundamentals of Web Development



The Core Process of the Web: HTTP

One of the most commonly used services on the Internet is the World Wide Web (WWW). The application protocol that makes the web work is **Hypertext Transfer Protocol** or **HTTP**.

Do not confuse this with the Hypertext Markup Language (HTML). HTML is the language used to write web pages. HTTP is the protocol that clients and web servers use to communicate with each other over the Internet. It is an application level protocol because it sits on top of the TCP layer in the protocol stack and is used by specific applications to talk to one another. In this case the applications are web browsers and web servers.

The Core Process of the Web: HTTP

HTTP is a connectionless text-based protocol. Clients (usually web browsers, but not limited to just web browsers) send requests to web servers for web elements such as web pages, assets and data.

After the request is serviced by a server, the connection between client and server across the Internet is disconnected. A new connection must be made for each request. Most protocols are connection oriented. This means that the two computers communicating with each other keep the connection open over the Internet. HTTP does not however. Before an HTTP request can be made by a client, a new connection must be made to the server.

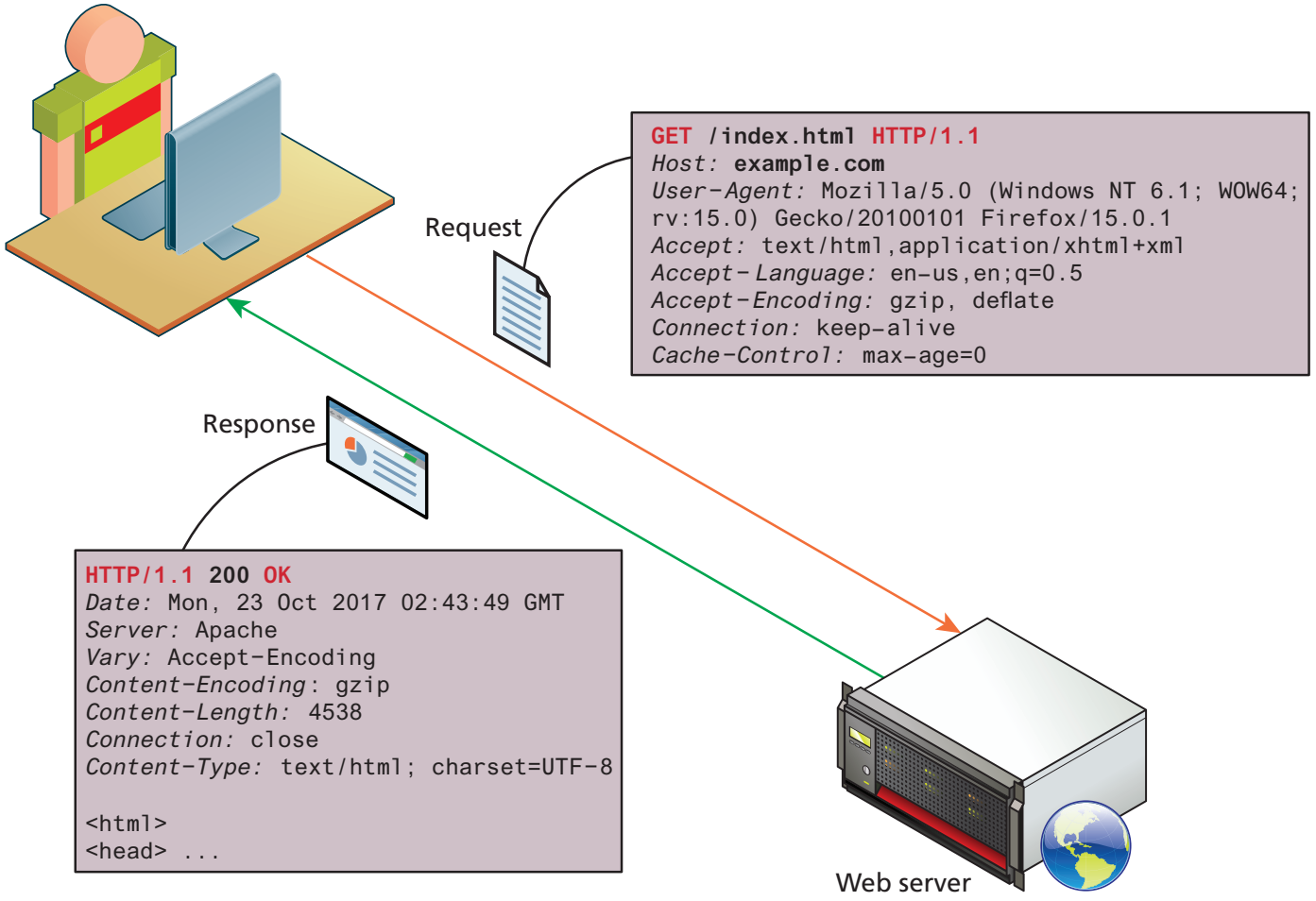
The Core Process of the Web

At the end of the day, the web is all about the communication of ideas. Everything in the web can be seen as a request and a response

- When you go to a news website, you're requesting news and receiving news in response.
- When you go to a shopping website, you're requesting product information and receiving relevant information.
- When your server receives input, it determines what to do with that input and outputs the proper response.

Your duty as a web developer is to make that communication possible. Your programs will get a request and give a response and allow that communication to occur as smoothly as possible.

The Core Process of the Web



The Request

Every time you navigate to a website, your browser sends a request on your behalf to the server.

- There is a lot to an HTTP request! https://developer.mozilla.org/en-US/docs/Web/HTTP
- Each request follows a standardized process! https://www.w3.org/wiki/How_does_the_Internet_work

Every request is formatted in a specific way, with the same data provided on each request. A request to http://google.com/ would look like:

```
GET /?gws_rd=ssl HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:40.0) Gecko/20100101 Firefox/40.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Cookie: PREF=ID=1111111111111111:FF=0:TM=1441084937:LM=1441084937:V=1:S=DKD8wPI-NAGQztx5; NID=71=0zCx
Connection: keep-alive
Cache-Control: max-age=0
```

Parts of That Request

The HTTP request has a lot of data that is sent including:

- HTTP verb signifying what action you are trying to (**GET**, **POST PUT**, **DELETE**).
- The protocol.
- The server you want to connect to (the HOST).
- The User-Agent: Information about the user's browser, platform etc..
- The location of the resource you want to access on that server (the location).
- Headers: headers are metadata about your request, these include cookies!.
- Request body (if there is one).

Parts of That Request

GET Request:

```
GET /?gws_rd=ssl HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:40.0) Gecko/20100101 Firefox/40.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Cookie: PREF=ID=1111111111111111:FF=0:TM=1441084937:LM=1441084937:V=1:S=DKD8wPI-NAGQztx5; NID=71=0zCx
Connection: keep-alive
Cache-Control: max-age=0
```

POST Request:

```
POST / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh;… )… Firefox/51.0
Accept: text/html,application/xhtml+xml,…,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-12656974
Content-Length: 345

-12656974
(more data)
```

Request headers

General headers

Entity headers

What Does the Server Do with That Information?

The server reads that request and determines what needs to be done in order to generate a response that makes sense for the data that the server has been given.

The server uses data sent in the request in order to generate a response. Some common types of data in a request that servers use are:

- Querystring parameters
- Headers
- Cookies
- Request body

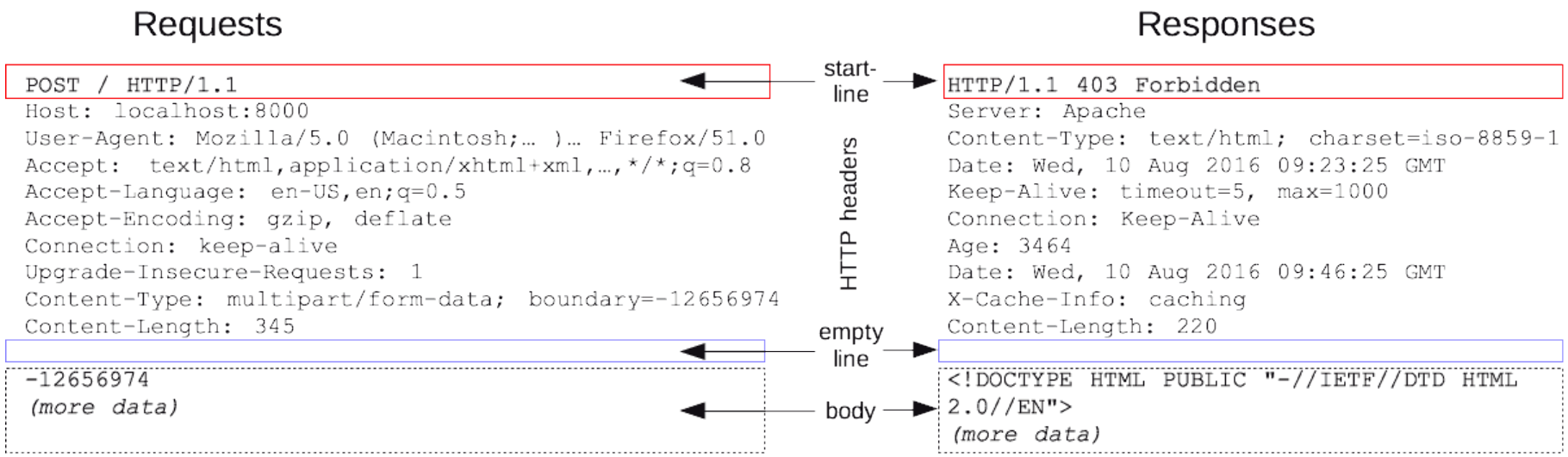
The Response

The server sends back a response that is similar to the request. It contains:

- A status code
  - Indicates whether or not the operation succeeded
- A set of headers
  - Cookies, data about the response such as content type
  - This is often "meta-data" about the response.
- Some form of response body:
  - An HTML Document
  - A JSON response
  - A File Stream
  - Plain Text
  - Etc.

```
HTTP/2.0 200 OK
Cache-Control: private, max-age=0
Content-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Date: Tue, 01 Sep 2015 05:31:36 GMT
Expires: -1
Server: gws
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
X-Firefox-Spdy: h2
```

Request/Response Example



Status Codes

Each response must return a status code indicating whether the request was successful, and a description is often included with each of the status code

Status codes in the...

- 200-299 range indicate a successful operation
- 300-399 range indicate some sort of redirection must occur
- 400-499 range indicate an error was made by the client during the request
- 500-599 range indicate some sort of error occurred on the server

You will use different status codes to describe different errors in this course Some status codes:

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

Express

Express is a very popular node package that is distributed on NPM which allows you to configure and run an entire web server.

- http://expressjs.com/

Express allows you to configure different routes and how they should compose a response.

Essentially, by using the Express Node module, you will use code to configure a server that will listen to requests and send out responses.

- This is all a web server is! It's not magic, it's just something that takes in requests and sends back responses!

What is a Route?

Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).

- http://expressjs.com/en/starter/basic-routing.html

For your web applications, you will configure many routes that will each perform some action.

By configuring routes, you can have the same URL perform different tasks based on having different request methods (HTTP Verbs)

Request Methods

There are many different types of request methods. For this course, you will be using 5:

- **GET**
  - The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.
- **POST**
  - The POST method is used to submit an entity to the specified resource, often causing a change in state or side effects on the server.
- **PUT**
  - The PUT method replaces all current representations of the target resource with the request payload.
- **PATCH**
  - Similar to PUT, but you can replace portions of the resource instead of the whole resource.
- **DELETE**
  - The DELETE method deletes the specified resource.
- Other Methods
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

GET Requests

Let us use a blog as an example.

GET requests are made when the client is requesting the representation of a resource

Location	Client is requesting…	Server is responding with…
http://myblog.com	The homepage of the blog	An HTML document with the most recent blog posts perhaps
http://myblog.com/post/2	The blog post with an ID of 2	An HTML document with the contents of the blog post
http://myblog.com/editor	A page with an editor to write new posts	An HTML document with a form so the user can compose a new blog post
http://myblog.com/post/2.json	A blog post with and ID of 2, but only the raw JSON data of the content	A JSON document representing the content of the blog post

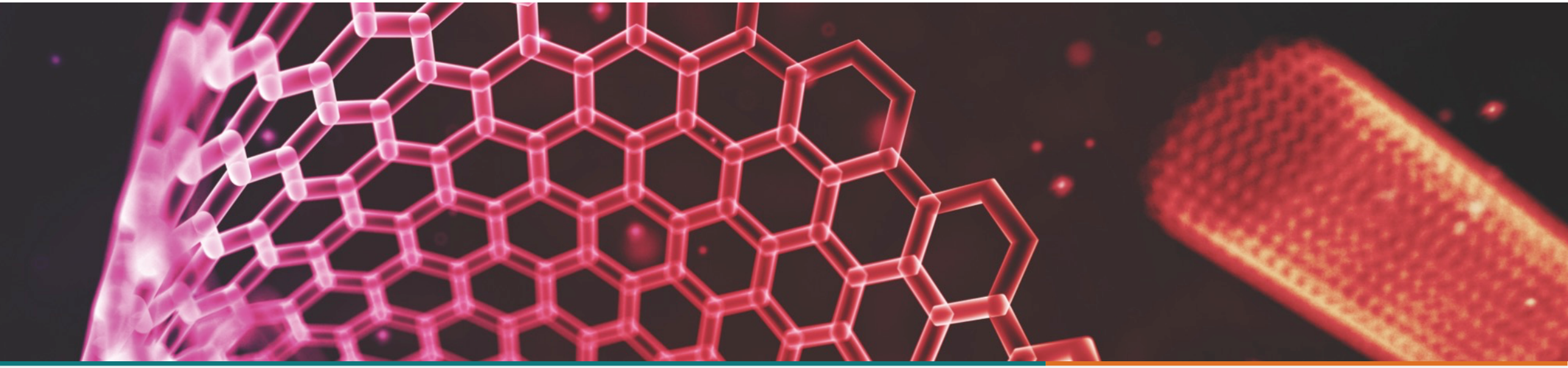
POST Requests

POST requests are made when the client is requesting to create some form of resource.

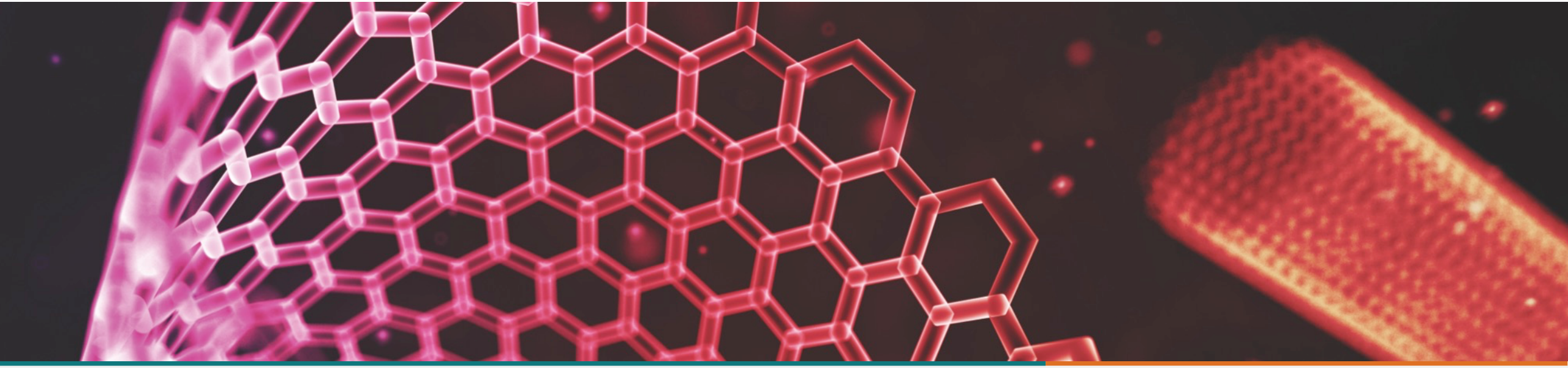
POST data comes with a message body that describes the content (which usually comes from an HTML form the user filled out, but not always)

Location	Client is requesting…	Server is responding with…
http://myblog.com/post	The data from the form that was filled out in the editor is submitted to this route for processing	An HTML document with the newly created post

Fundamentals of HTML and CSS



HTML



Creating an HTML Document

HTML (Hyper Text Markup Language) is a markup language; it is a way of describing content. A file written in HTML is referred to as an HTML document.

Our first HTML documents will exist on our desktops, rather than on a server.

- HTML documents are simply text files that are formed following the HTML standard.
- HTML is composed of a series of tags to describe the content.
- An HTML document is a text document that describes a **web page**.

Your browser will interpret this document and render it!

What's in an HTML Document?

- An HTML has a series of elements
  - Open tag plus attributes and properties
  - Nested elements
- Some very important
  - HTML Doctype
  - HTML Element
  - Head Element
  - Body Element
- Elements can be identified by an ID: ID can only be used once per document
- Elements can identify a group by their class
- Elements are described in the document and rendered in the DOM

The Browser is Only Half the Battle

The web isn't just accessible via a browser. As modern web developers, we must care about:

- Screen readers
- Search Engine Crawlers / Other AI

There is a growing movement to make the web more accessible

- Leveraging HTML's strengths
  - Navs in the nav
  - Labels in forms
  - Using headings properly
  - Attributes to help screen readers
- Making designs accessible
- Tables for tabular data only
- Make sure you can navigate via keyboard

Separating Style and Content

Before we can think in terms of organizing our data meaningfully, we need to understand what HTML does not accomplish; the way your document looks.

- **Elements are used to describe your data; CSS is used to style your data.**
- While browsers give many native styles to elements by default, elements are not inherently used for styling. This is why tags for bolding and italicizing text, or changing fonts, were deprecated in HTML5.
- There needs to be a clear separation between style and content; any overlap is a happy coincidence.

While writing HTML, thinking in terms of content first, then styling often leads to more logical, and easier to style documents.

**You will get points off your labs if you use tags like <b>, <i>, **

Types of Text

Across the web, text is used to portray many different types of things.

- Headings / Titles in your content (**h1**, **h2**, **h3**, **h4**, **h5**, **h6**)
- Regular paragraph (**p**)
- Generic groups of text / adding custom definitions or functionality to text (**span**)
- Emphasized text (**em**)
- Important text (**strong**)
- Addresses (**addr**)
- Citations (**cite**)
- Abbreviations (**abbr**)
- Quotes (**blockquote**)

Using the right kind of element to describe text is very important for SEO, non-browser accessibility, and readable code.

- Even without different styles, those tags help readers understand their document.

The Layout of Your Content

There are many elements that describe the layout of your content

- How to navigate content / your document (**nav**)
- Grouping your content into sections that have something to do with each other (**main**, **section**)
- Denoting a header for content or your document (**header**)
- Denoting a footer for content or your document (**footer**)
- Grouping content into a self-contained article (**article**)
- Stating that certain content is secondary (**aside**)
- Grouping divisions of content (**div**)

Validating HTML

For this course, the validity of your HTML is highly important.

Having valid HTML means your browser does not have to guess how to fix it, which can lead to drastically wrong web pages and pages that cannot be made sense of.

The w3 website has an easy to use validation service that tells you issues and proposed solutions:

- https://validator.w3.org/#validate_by_input

You should view the source of your page, copy, and paste it all into the HTML validator's 'direct input' section before submitting HTML in this class.

You should strive to write as perfect HTML as possible.

You will get points off your labs if you have ANY invalid HTML

Attributes and Properties

Elements can have many classes and properties attached to them to further describe them. The difference between the two of those are nuanced and deals with the state of the page.

- This is an example of how browsers had to adapt to a set of standards that were not always fully thought out.

Attributes appear in key-value fashion when writing HTML:

- `<a href="http://google.com">Go To Google</a>`

The ***href*** attribute is set to Google's home page

Elements are parsed and, as they are changed, keep track of the set of properties. Some properties come from their attributes; others come from user input.

Classes and IDs

Elements will often be described with classes and IDs to signify them in some way. Many elements can share a class, and each element can have many classes

```
<div class="panel panel-default"></div>
```

Has two classes, panel and panel-default

```
<div class="panel panel-danger"></div>
```

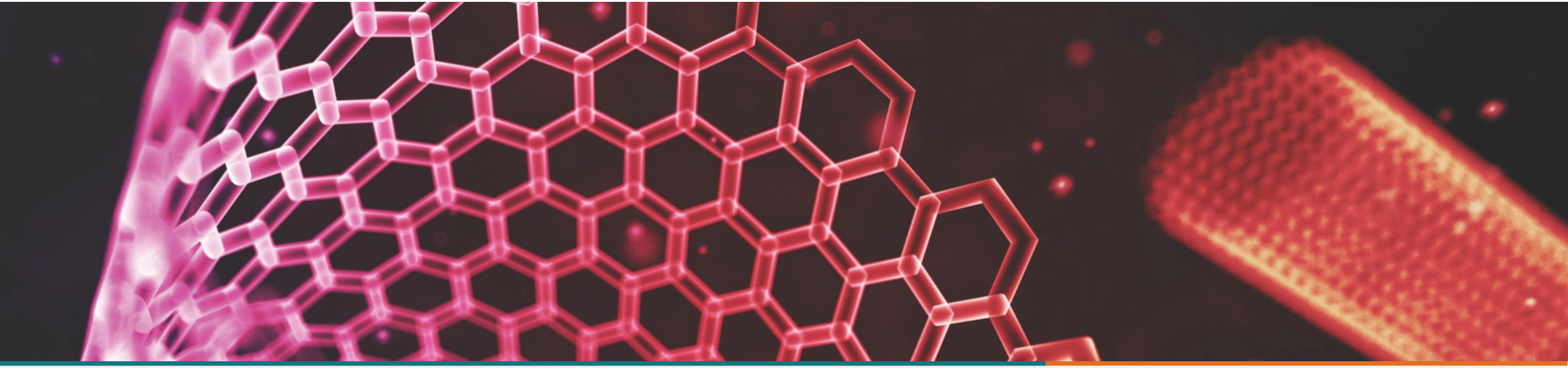
Has two classes, panel and panel-danger

However, only one element can have a particular ID:

```
<div id="about-me"></div>
```

Classes and IDs are most often used to style elements and target elements with JavaScript to add functionality.

CSS



What is CSS?

CSS is the language that we style HTML documents with.

- Cascading
- Style
- Sheets

CSS allows you to define rule-sets, which are selectors (identifiers that target HTML Elements) and rules (rules that define visual properties)

You can use CSS to describe how a document is presented in different contexts

- In a browser
- On a projector
- When printed

It allows you to take your documents and make them look like fully designed pages.

Adding CSS

It is very easy to add a CSS stylesheet to your HTML document.

In the head element, you add a link element like such:

```
<link rel="stylesheet" href="/public/site.css">
```

How Does CSS Work?

CSS is a simple language based on only three pieces of data:

- A selector, which is a pattern that will match elements
- A declaration, which has:
  - A property, which defines what property you will update
  - A value, which defines what you want to do to that property.

A set of selectors and a set of declarations makes a rule-set

Each rule-set can have multiple selectors, and multiple declarations.

CSS Syntax/Example Rule-Sets

```
h2 {
  text-align: center;
}

p.bio, .about-me .career-info {
  font-size: 16pt;
  border-bottom: 1px solid #333;
}
```

The first rule-set will target all h2 elements in the document and center all the text contained inside the element.

The second rule-set will target all p tags that have a class of bio, as well as all elements with a class of career-info that are contained inside an element that has a class of about-me; there can be any number of elements and nested layers of elements between about-me and career-info. Any matching elements will have a bottom-border that is 1 pixel in size, grey colored, and a solid line; they will also have their font set to be 16pt.

Selectors

CSS	Name	Selects
`*`	Universal	Any and every element
`div`	Element	All div elements
`.foo { }`	Class	All elements with a class of *foo*
`#bar { }`	ID	The single element with the id of *bar*
`#bar .foo { }`	Descendant	All *.foo* that are contained inside *#bar*
`.parent > .child`	Child	All .child directly inside of .parent
`.post + .divider`	Adjacent Sibling	All .divider that are directly after .post in their parent.
`.post ~ .subtext`	General Sibling	All .subtext that come anywhere after .post in their parent.
`.post:pseudoclass`	Pseudo-Class	All .post that have a particular pseudo class.
`[role='navigation']`	Attribute	All elements that have an attribute named 'role' with the value of 'navigation'. You can do this for any attribute and value.

Pseudo Classes (All Types)

Pseudo Class	Selects
`:first-child`	The first child of a parent
`:last-child`	The last child of a parent
`:first-of-type`	The first element of a type, inside of a parent; does not accept a class or id.
`:last-of-type`	The last element of a type, inside of a parent; does not accept a class or id.
`:nth-child(an+b)`	Selects all .child directly inside of .parent The formula (*an+b*) describes which elements are targeted; elements start at index 0. (*2n+1*) would select elements at index 1, 3, 5, 7 while (*3n*) matches at index 0, 3, etc.
`:nth-last-child`	Same as :nth-child, except starts from the last element
`:nth-of-type`	As as :nth-child, except works with element types
`:empty`	An element that has no content, including whitespace; can have comments.
`:target`	The element matching the target of your current hash.

Size Units

Unit	Name	Description
`px`	Pixel	Size of a pixel on the screen
`mm`	Millimeter	Size of a millimeter
`cm`	Centimeter	Size of a centimeter
`in`	Inch	Size of 1 Inch; generally, 96 pixels per inch
`pt`	Point	Size of 1/72 inch
`pc`	Pica	Size of 12 Points
`em`	em	Calculates the size based on the elements font size, or parent's font size. Literally, relative to the "M" in a font at a size.
`%`	Percent	Calculates the percentage of a size relative to a property on the parent element.
`rem`	Rem’s	Same as em, but based on the root element. Allows for very responsive layouts just by updating the root element.

Color Units

Color	Unit Name	Description
`rgb(255, 255, 255)`	RGB	Allows you to describe colors as how red, green, and blue they are from 0 to 255 each. Allows for 16,581,375 colors.
`rgba(0, 192, 45, .5)`	RGB Alpha	Same as hex, where the last decimal is transparency from 0 (clear) to 1.0 (solid)
`#A90BEF`	Hex	A hex triplet; has 2 hex digits representing each color (Red, Green, Blue). Describes your color from Allows for 16,777,216
`hsl(0, 20%, 50%)`	Hue, Saturation, Lightness	Takes the hue from 0 to 360; 0/360 are red, 120 green, 240 blue. Saturation determines what percent of that color you use. Lightness is a mask over saturation, adding a white layer. 50% is the normal value.
`hsla(0, 20%, 50%, .5)`	Hue, Saturation, Lightness, Alpha	Same as above, where the last value is transparency.

Text Rules

Rule Name	Example	Outcome
font-family	`font-family: "Open Sans", "Helvetica", sans-serif;`	If the user has Open Sans font, will use open sans; else Helvetica; else sans-serif (generic font).
font-size	`font-size: 18pt;`	Sets font size at 18pt (about ¼ of an inch)
line-height	`line-height: 3;`	Will make each line have a height of 3x font size.
text-decoration	`text-decoration: underline;`	Text will be underlined.
font-weight	`font-weight: 700;`	Font will be bold; norm is 300.
text-align	`text-align: center;`	Text will be centered inside parent element.
font-variant	`font-variant: small-caps;`	Makes an element use small capitalized letters for lowercase.
font-style	`font-style: italic;`	Makes text italicized.
font	`font: 2em "Open Sans", sans-serif;`	Sets font at 2em large, Open Sans (fallback to sans-serif)
font-family	`font-family: "Open Sans", "Helvetica", sans-serif;`	If the user has Open Sans font, will use open sans; else Helvetica; else sans-serif (generic font).

Color and Background

Rule Name	Example	Outcome
color	`color: #690;`	Changes text color to a green.
background-color	`background-color: #000;`	Sets background color to black.
background-image	`background-image: url(gradient.png);`	Uses gradient.png as background for element.
background-attachment	`background-attachment: fixed;`	Makes the background image fixed in browser.
background-clip	`background-clip: border-box;`	Sets the background to fill entire box model.
background-position	`background-position: left center;`	Positions your background anchored to the left horizontally and center vertically.
background-repeat	`background-repeat: repeat-x;`	Sets background to only repeat horizontally.
background-size	`background-size: contain;`	Scales the background image to fit length or width and letterboxes.
background	`url(gradient.png) no-repeat;`	Shorthand for multiple properties.
color	`color: #690;`	Changes text color to a green.

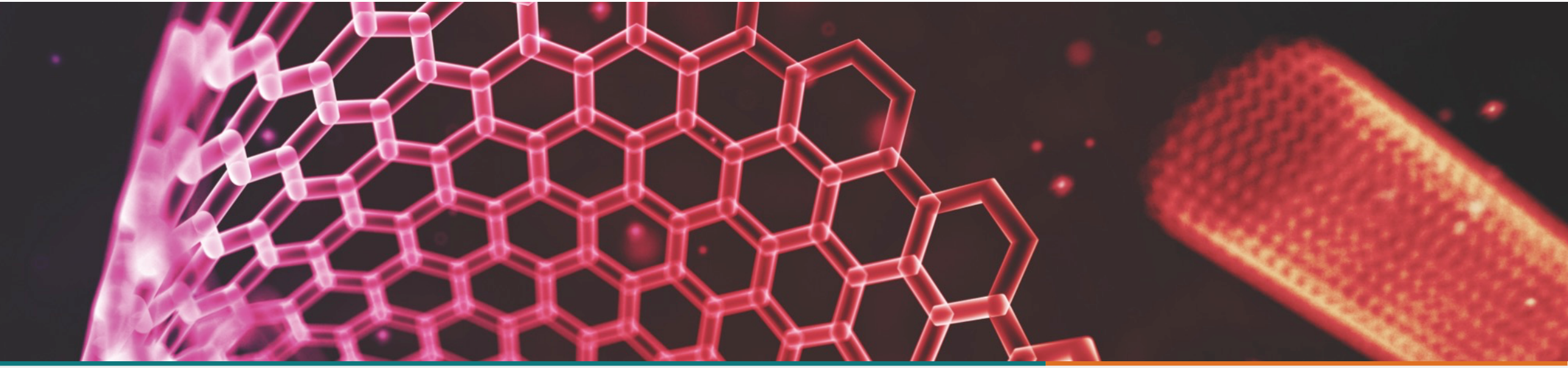
Setting a Border

Example	Outcome
`border-left: 1px solid #999;`	Sets a 1px wide solid border on the left; grey.
`border-right: 5px dashed #558abb;`	Sets a 5px wide dashed border on the right; light blue.
`border-top: 1px solid #999;`	Sets a 1px wide solid border on the top of the element; grey.
`border-bottom: 1px solid #999;`	Sets a 1px wide solid border on the bottom of the element; grey.
`border: 2px dotted #558abb;`	Sets a 2px wide dotted border on all sides; light blue.

Putting it All Together

Example	Outcome
`box-size: border-box;`	Makes your element include padding and the border in width.
`padding: 5px;`	Adds 5 pixels of padding on the element.
`margin: 0px auto;`	No margins on top or bottom; left and right will automatically be calculated, possible centering element in parent.
`width: 75%;`	Sets width of element to be that of 75% of the parent's width.
`height: auto;`	Height will automatically be determined by content;
`max-width: 500px;`	Element will not exceed 500px wide.
`position: relative;`	Element will be moved relative to static position.
`top: 10px;`	Element will be 10px below where it was originally meant to be

Basic Web Accessibility



What is Accessibility?

As web developers, we have the opportunity to make sure that our websites and web applications are consumable by people with a number of disabilities.

Even simple pages can have a number of issues that cause a person with some form of disability to be unable to fully use it; a form without labels, for example, is much harder for a screen reader to parse. A visually impaired user would struggle.

Even your design affects web accessibility: a lack of color contrast can make text nearly invisible to some of your users!

Testing Accessibility

There are many ways we can test for accessibility, but to start, we will be using the tota11y tool

- http://khan.github.io/tota11y/

The tota11y tool is an accessibility visualizer that can be installed via a bookmarklet.

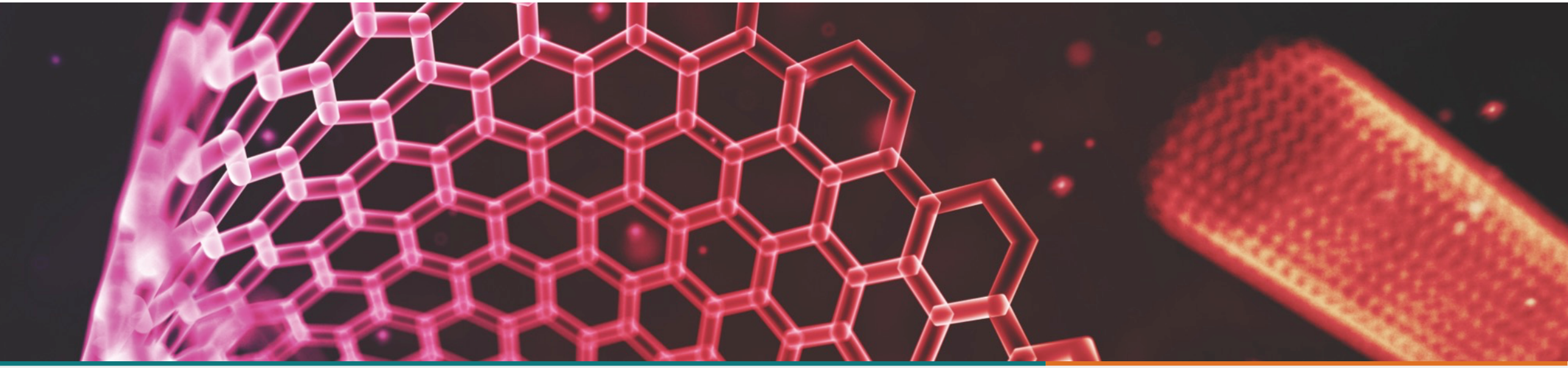
This tool will allow you to identify how assistive technologies would interpret your website.

Going Forward

Going forward, all HTML submitted must pass tota11y tests.

Points will be deducted for labs and final project components that fail accessibility checks.

AJAX



What is AJAX?

AJAX (asynchronous JavaScript and XML) is a series of techniques used to have clients execute JavaScript code in order to request resources without leaving their current page.

This means that you can write code that makes network requests on the client's behalf to access data on your server.

We will be using jQuery to perform our AJAX requests.

What Does Asynchronous Mean?

AJAX calls are inherently asynchronous; you do not know when they will complete, so they pass data through callbacks.

- The request is made on a background thread that does not block the UI
- This request can finish anytime after it is sent, so you want to hook into your event listener before the request is sent; a request could theoretically complete before you listen for the response.
- The request may not ever complete successfully

Requesting JSON Data

With Express, having a route return JSON is very easy.

You can easily request and use JSON data by:

- Make an AJAX request to a route that sends JSON in its response
- Listen for a callback state change
- Check the data to see what you should do (ie, was it successful? Then render; otherwise, show error)
- Manipulate and use the data as needed

Submitting JSON Data

We can post JSON, as well! This is particularly useful, as you can finally start sending numbers and booleans as well as strings by sending JSON.

In order to POST data in JSON format, you format your request and pass JSON in the body of the request.

Using the bodyparser middleware allows you to have this JSON easily parsed into our request body field in our Express routes and middlewares.

Benefits of Using AJAX

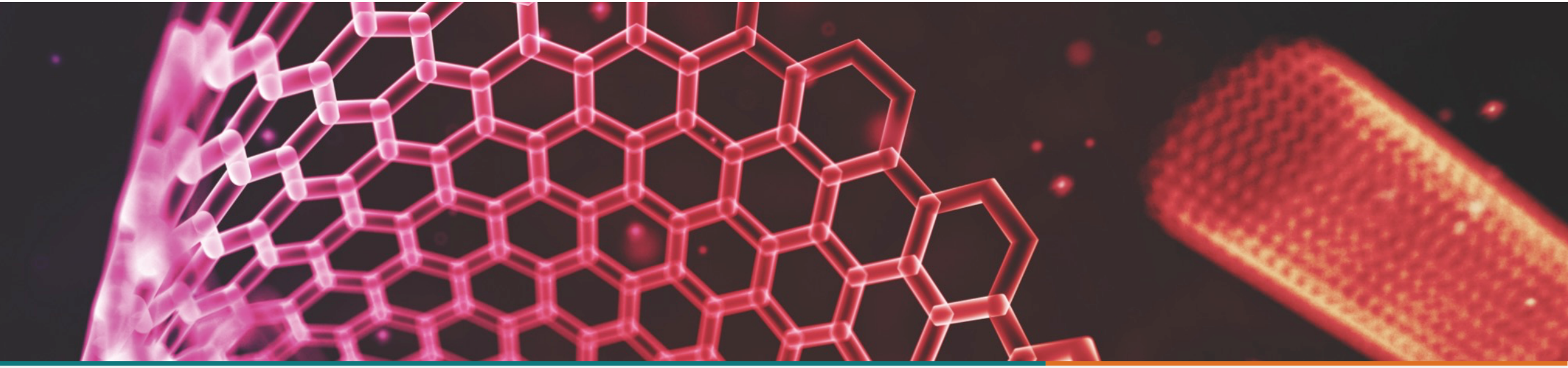
Smaller payloads; you can only send down data that the user cares about.

You can split up your code into a more modular setup.

You can keep updating one page, rather than re-requesting entirely new pages all the time!

- Since one page is updating, you will not have to re-request and re-render the same resources such as stylesheets and JS files, making your application often perform much better.

Middleware



What is Middleware?

A middleware is a function that has access to the request and response objects. These functions can:

- Execute any code.
- Make changes to the request and the response objects.
- End the request-response cycle.
- Call the next middleware function in the stack.

You can apply middleware to the entire application, or portions of the application

- You can apply it to a portion of the application by supplying a path as the first parameter to the middleware function

It is called a middleware because it sits in the middle of the response and the request.

Practical Uses for Middleware

Middlewares are useful for a number of reasons, and have many common uses such as:

- Logging requests
- Authentication
- Access control
- Caching data
- Serialization

Writing a Middleware

Writing a middleware is extremely easy

- Register your middleware, optionally providing a path to apply that middleware to
- Have your middleware perform a task and when done:
  - Have your middleware end the response
  - Have your middleware call the next middleware

As an example, see the ***app.js*** file, which has several middleware functions:

- One which will count the number of requests made to your website
- One which will count the number of requests that have been made to the current path
- One which will log the last time the user has made a request and store it in a cookie.
- One which will deny all users access to the ***/admin*** path.

What is Authentication?

Authentication is the process of verifying what user is currently operating in a system. You would be most familiar with it through the use of usernames and passwords.

For example, on MyStevens, you are authenticated by supplying your Stevens username and password. By providing this data, the system sends a Cookie to your browser that stores a session ID that associates your requests to your user account.

What is Authorization?

Authorization and authentication are often confused. While **authentication** handles who you are, **authorization** is the process of validating what you can access.

For example, as a faculty member, my user account is authorized to input grades to student accounts, whereas student accounts are not.

- Even more granular, I am authorized in the system for CS-546 grades, but not MGT-671

Authorization comes in many forms; typically, you will see three or more layers of authorization

- Public facing pages; even users that are not authenticated can see these pages. For example, your homepage or login page would be public facing
- Authenticated only pages; pages that all authenticated users can see
- Role or claim based pages; pages that users can only see if they have certain account types, such as faculty being able to access grading features whereas students cannot

Authentication

Authentication is the act of confirming the identity of a person, group, or entity. In web technology, this often means creating a **user login system**

- You will use a combination of data in order to identify a user.

There are many other forms of authentication in web technology:

- You can make an authentication system that allows you to limit API Access
  - Force users to have a token
  - Allow users a certain number of access hits a month
- You can selectively allow or dis-allow access to resources based on user login state

Implementing Authentication

In order to implement authentication and create a user login system, we will be breaking down the task into several steps:

- Creating and storing users
- Allowing users to login via a form
- Storing session data in a cookie
- Validating the data stored in the cookie
- Storing the user as part of the request object.

Let’s walk through this process.

Creating and Storing Users

The first step of authentication is very, very easy; you have to create, and store users.

There are some things you'll be storing, and some things you'll be storing in a very specific way

- First off, you will never store a plaintext password. You will be using the bcrypt package in order to create a hash of the password
- For the sake of authentication, you're going to be adding an array for users that will keep track of multiple session identifiers. These session identifiers will allow you to keep track of logged in browser sessions

You will need to create a form to allow users to signup, where you will need to check for:

- Duplicate username/emails/other non-duplicatable data
- Mixed case of username/email addresses
  - patrickhill@stevens.edu should be treated the same as PaTriCkHill@stevens.edu and PATRICKHILL@steVens.edu same for usernames: phill, PHILL, Phill should be treated as equal.
- Existence of passwords

Allowing Users to Login via a Form

This step is extremely easy!

You will need to provide users with some way to actually perform a login. You will need to setup a form that allows users to **POST** their username and password to a route.

This route will need to validate the username and password provided against entries in the database.

- You will retrieve the user with that matching username
- You will use ***bcrypt*** to compare if their supplied password is a match to their hashed password that is stored in the database
  - I have created a simple file, ***bcrypt_example.js*** to demonstrate how to use ***bcrypt*** to create and compare hashes.

If there is a match, you can proceed; if not, you will simply not allow the request to continue and display an error to the user.

Storing Session Data in a Cookie

If the user logged in with proper credentials, you will then create a session id! This session ID should be some sort of very long identifier, such as a GUID.

Rather than storing the user ID or username, and password in cookies, we instead are opting to store a session ID so that the username or password cannot be intercepted.

This session ID will be passed to the user via-cookie and will also be stored as one of many session ID's on the user in the database. So when the user logins in, the server can generate the GUID, store that in an array of session ID's in the user document and send that back as a cookie to the client in response.

Validating the Data Stored in the Cookie

It is now time to make your middleware!

Your middleware should run on each request, and will check for a cookie containing a session ID

- If it contains a session ID, you will check the database for a single user that has that session ID stored in their session ID field
  - If there is a match, you've found your user!
  - If not, your request is coming from an unauthenticated source; expire their cookie.

- If not, your request is coming from an unauthenticated source.

Storing the User in the Request Object

In your middleware, you have access to the request and the response objects, and you can add properties to them easily.

If you are able to associate a session ID with a user, you may define a property on the request (or response!) object that stores the user, or some representation of them (ie: just storing the user ID).

The data you store will be accessible:

- In middleware that are defined after the authentication middleware
- In your routes

If you define middleware after your authentication middleware, you can attach them to particular paths (such as ***/user***) and, if a user is not logged in, you can redirect them. You can also do things such as check on other paths (ie: ***/admin***) to see if the user has permission to access those paths, and redirect if not.

Logging Out

Logging out is extremely easy, and only has two steps!

- After hitting a logout route, you will expire the cookie for the session ID
- You will remove the session ID from the user's session ID list
- You will invalidate any other cookies that are relevant to the user.

By doing both of those, you will have successfully invalidated the session and the user will no longer be authenticated.