**🎤 Speaker Notes – Slide 1: Title Slide**

“Good [morning/afternoon], everyone. My name is **Lan Nguyen**, and I’m excited to share a short sample lecture from **GEOG 4105 – Geospatial Web Programming**.”

“The title of this lecture is **‘From Code to Map: Using JavaScript to Power Interactive Web GIS.’** That’s exactly what we’re going to explore — how JavaScript, the language of the browser, helps us build rich, interactive maps that users can click, zoom, and explore.”

“This is a sample from **Week 3** of the course, which is where we introduce students to JavaScript and how it ties into building functional geospatial web apps.”

“Whether you're already comfortable with GIS or just starting to explore the web development side, this lecture is meant to connect spatial thinking with coding — giving you the tools to bring maps to life in the browser.”

“Let’s dive in.”

**🎤 Speaker Notes – Slide 2: Learning Objectives**

“Today’s session is all about giving you a hands-on introduction to **JavaScript as the engine behind interactive web maps**. By the end of this short lecture, you’ll have a solid grasp of how JavaScript is used in modern GIS apps and how to connect it to real spatial data.”

1. **Understand how JavaScript is used in modern web GIS**  
   “JavaScript is the language that runs in your browser and allows users to click, interact, and explore maps. It’s the glue that connects your spatial data to visual, interactive elements.”
2. **Learn core JavaScript concepts relevant to GIS**  
   “We won’t cover the whole language, just the parts most useful for GIS: variables, functions, how to grab HTML elements, and how to react to user input — like clicking a feature on a map.”
3. **Load and visualize GeoJSON data on a web map**  
   “GeoJSON is the most common format for spatial vector data on the web. You’ll see how to fetch that data and render it dynamically on a Leaflet map using just a few lines of JavaScript.”
4. **Trigger map interactions using JavaScript events**  
   “Interactivity is what makes a GIS map powerful — popups, tooltips, filters, zoom. Today we’ll learn how to connect user actions (like clicking on a point) to JavaScript functions.”

“These objectives build a foundation you’ll need for next week’s work — including deeper interaction, styling, and real-world data. Think of this as unlocking the ability to make your maps respond and communicate!”

**🎤 Speaker Notes – Slide 3: Why JavaScript for GIS?**

“So, why JavaScript? Why do we focus on this language in a GIS course? The answer is simple: **JavaScript is what powers web interactivity.** If you’ve ever clicked a marker, hovered to see a popup, or filtered map data on a dashboard — JavaScript made that happen.”

“First, JavaScript runs directly in the **web browser**, which means no installs or plugins. It’s the native language of the web and controls how content behaves on the page — including maps.”

“Second, the **most popular web mapping libraries** are all built with JavaScript — things like **Leaflet, Mapbox GL JS**, and **OpenLayers**. They allow us to load basemaps, overlay GeoJSON, add interactivity, and more — all using JavaScript.”

“Third, JavaScript supports **real-time and event-driven behavior**. That means we can fetch new data on the fly, respond when users click a feature, or change the map when filters are applied. This makes our web maps dynamic and user-responsive.”

“And finally, JavaScript is an **essential job skill**. If you want to work as a Web GIS Developer, a Frontend Analyst, or even in environmental informatics, you’ll need JavaScript. It’s the bridge between raw data and meaningful user experience.”

“So today’s goal is to get you comfortable writing and reading JavaScript code — especially the pieces that help you load data, handle events, and bring your map to life.”

**🎤 Speaker Notes – Slide 4: JavaScript in a Web Map Stack**

“This slide gives us the big picture — the architecture of a typical GIS web application. Think of this as the **system stack** behind any interactive map you see online.”

“At the top, we have the **Frontend** — made up of three key technologies:  
HTML provides the structure of the page,  
CSS handles the look and feel,  
and JavaScript controls how the page and map behave.”

“Then we layer on a **mapping library**, in our case, **Leaflet.js**. Leaflet is a lightweight and widely-used JavaScript library for adding maps and spatial features to a webpage. It lets us do things like load a basemap, add popups, or draw polygons — with just a few lines of code.”

“Next is our **data format**: GeoJSON. This is a standard for encoding geographic features like points, lines, and polygons. It’s simple, lightweight, and easy to use with JavaScript. Most web-based spatial data today is served as GeoJSON.”

“Finally, we can connect all of this to a **backend** — like a Python Flask app — that serves GeoJSON data or interacts with databases. This becomes important when you want to host your own datasets or expose APIs to users.”

Backend is optional because many web GIS applications can run entirely in the browser using static files (HTML, CSS, JavaScript, and GeoJSON data). For simple maps or demos, you can load local or public GeoJSON files directly without a server.

A backend (like Flask or an API) is only needed if you want to:

* Serve dynamic or private data,
* Connect to a database,
* Perform server-side processing,
* Enable user authentication or data updates.

“Altogether, this stack shows how **client-side JavaScript** connects the user, the data, and the map interface. And the best part? Much of this can run entirely in the browser — which is why it’s so accessible and scalable.”

“In the rest of the lecture, we’ll see how to build and connect these layers step-by-step, starting with the Leaflet map.”

**🎤 Speaker Notes – Slide 5: JavaScript Building Blocks**

“This is where we start getting into the JavaScript language itself. Don’t worry — we’re not diving into every syntax rule or obscure feature. We’re focusing on the parts of JavaScript that are most useful for building interactive maps.”

“First up, **variables**. These are how we store and reference values in JavaScript. We use let or const to define them. For example, let mapCenter = [51.05, -114.07]; — this stores a coordinate pair.”

“Next are **functions** — reusable blocks of code that define behavior. You might write a function like zoomToFeature() to pan and zoom to a selected feature. Functions help you structure your logic and respond to map events.”

“**DOM manipulation** means interacting with the HTML elements on your page. JavaScript can grab elements like map divs or buttons using document.getElementById() or querySelector() — and change their content, style, or behavior.”

“Finally, we have **events** — like onClick, onMouseOver, and onLoad. These let us respond when a user clicks a marker or hovers over a polygon. Leaflet and the browser both provide ways to attach event listeners to elements or features.”

“This slide is all about the vocabulary of interactivity. These four concepts — variables, functions, DOM, and events — are what power every interactive experience on a map.”

**🎤 Speaker Notes – Slide 6: Introducing Leaflet.js**

“Now that we understand the JavaScript basics, let’s look at how they come together in Leaflet — a JavaScript library for web mapping.”

“Leaflet is incredibly popular because it’s lightweight, beginner-friendly, and works across devices. It’s perfect for embedding interactive maps on a website or web app.”

“This example code is how you initialize a Leaflet map. You target an HTML element by its ID — in this case, map — and set a view with coordinates and a zoom level.”

let map = L.map('map').setView([51.05, -114.07], 13);

“This centers the map on Calgary, for instance. Then you add a tile layer — here, we’re using OpenStreetMap.”

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png').addTo(map);

“This pulls in the visual basemap tiles from a public tile server and renders them on your page. These two lines are the foundation of any Leaflet project.”

**🎤 Speaker Notes – Slide 7: Adding Interactivity**

“Let’s add a marker to that map. This line creates a point at a specific coordinate, adds it to the map, and binds a popup to it.”

L.marker([51.05, -114.07]).addTo(map).bindPopup('Downtown Calgary').openPopup();

“The marker appears immediately, and the popup opens by default with our custom message.”

“You could let students play with this in their browser or a live editor. Try changing the coordinates to represent their hometown, or modify the message. They could even customize the icon or style using Leaflet options.”

“Markers and popups are often the first step in making your map interactive and meaningful — especially when working with points of interest.”

**🎤 Speaker Notes – Slide 8: Loading GeoJSON Data**

“Now let’s bring in real spatial data. GeoJSON is the go-to format for vector data on the web — it’s human-readable and plays well with JavaScript.”

“This example shows how to fetch a GeoJSON file using JavaScript’s fetch() API, then pass the data to Leaflet.”

javascript

CopyEdit

fetch('data/parks.geojson')

.then(res => res.json())

.then(data => {

L.geoJSON(data).addTo(map);

});

“The fetch() call loads the GeoJSON file from your project directory (or from an online source). We then parse the response as JSON, and Leaflet takes care of rendering the features — whether they’re points, lines, or polygons.”

“You could load parks, trails, community boundaries — anything with spatial attributes. This pattern is also useful for working with live APIs, such as an Open Data portal.”

**🎤 Speaker Notes – Slide 9: Live Coding Demo**

“This is where we put it all together in a live demo. I’ll walk through this step-by-step so you can see how these pieces connect.”

“We’ll:

* Create a Leaflet map,
* Add a marker,
* Load a GeoJSON file and style the features,
* And finally, make it interactive by adding popups or tooltips.”

“During the demo, I’ll explain what each line of code does and encourage you to follow along. This is your chance to ask questions and see JavaScript in action — not just in theory.”

“In a full class, I’d do this with my browser and VS Code side-by-side, so you can watch the code update and the map respond instantly.”