Client-Side Web GIS

JavaScript

Image from NASA: https://commons.wikimedia.org/wiki/File:Earth_Western_Hemisphere_transparent_background.png#filelinks

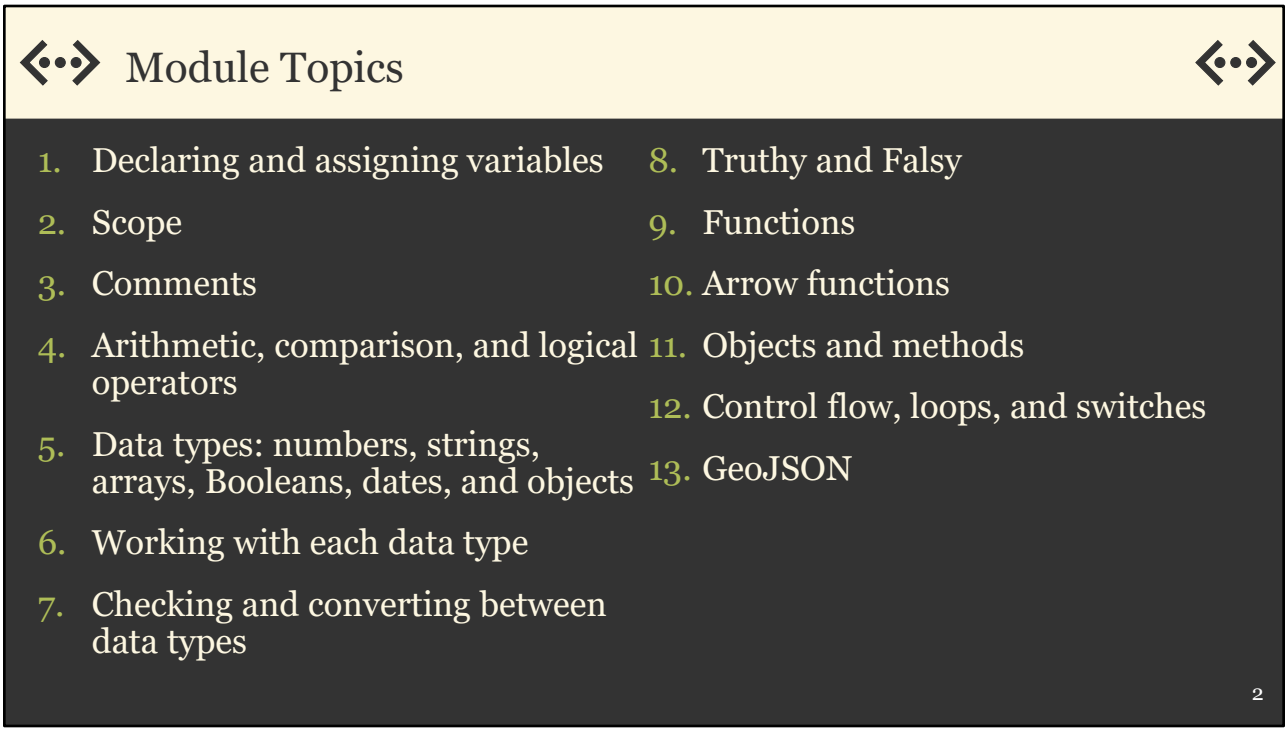
In the previous modules, we discussed defining web content using HTML and styling pages using CSS and Bootstrap or w3.css. These technologies do not offer many options for interactions on a webpage, other than clicking on links or using pseudo-selectors.

Website functionality and interactions are generally defined using JavaScript; further, web mapping APIs are generally built in JavaScript (for example, the ArcGIS Maps SDK for JavaScript and the Leaflet JavaScript API). So, for web development and web mapping JavaScript is an important skillset.

Note that JavaScript and Java are separate programming languages that are not related to each other. JavaScript is a general purpose, object-oriented coding language that has been adopted for client-side web programming.

In this module, we will also discuss GeoJSON, which offers a method to store geospatial data that can be read and manipulated with JavaScript.

This module will focus on the basics of the JavaScript language. A later module will explore how JavaScript is used to add interaction to webpages and to manipulate the document object model (DOM).

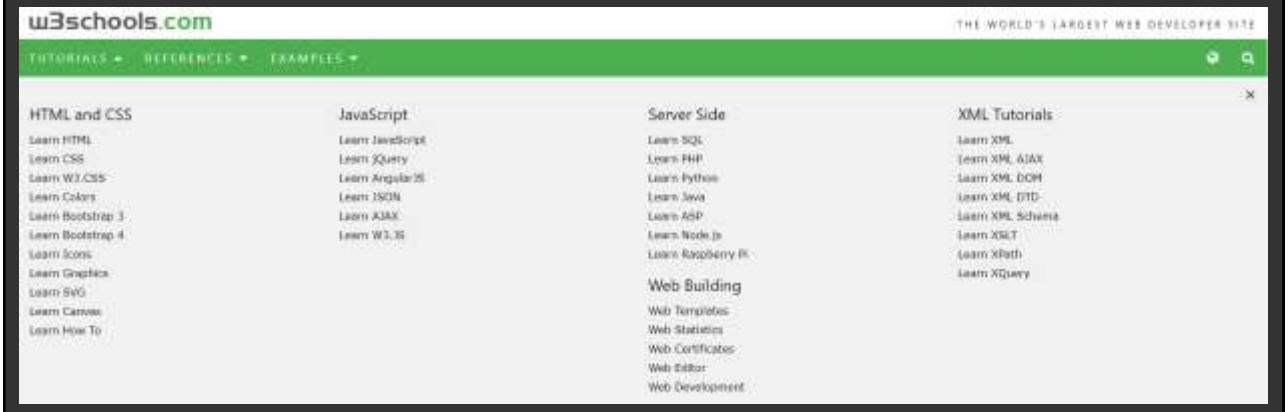


These are the topics covered in this module.

Preparation and Resources

3

W3schools

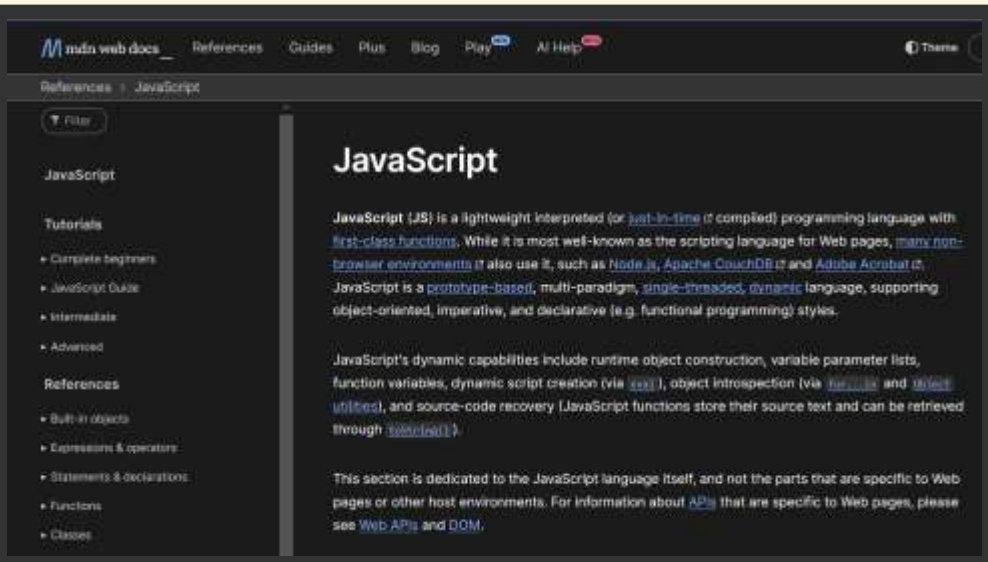


https://www.w3schools.com/default.asp

4

Additional JavaScript resources are available on the w3schools.com webpage.

Documentation



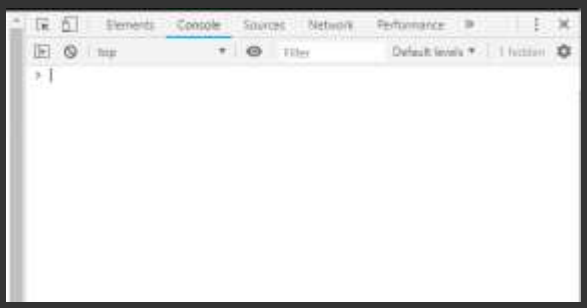
https://developer.mozilla.org/en-US/docs/Web/JavaScript

5

The Mozilla free software community, which also maintains the Firefox web browser, provides documentation for the JavaScript language.

Developer Tools

- Ctrl + Shift + I (Chrome or Firefox)
- Can execute JavaScript from the Console
- Can use for debugging



6

Where can you execute and experiment with JavaScript?

One option is to use the development tools associated with your web browser.

In this course, we will use the Developer Tools made available in Google Chrome or Firefox. Specifically, JavaScript can be executed using the Console.

Variables, Arithmetic, and Logic

Declaring and Assigning Variables

- Start with var, let, or const
- = is used for assignment
- End line with ;
- Can hold numbers or strings
- "" or '' around strings
- Can declare and assign in same statement
- Can declare then assign in a second line
- You can declare multiple variables in one statement

- Variable names:
  - must begin with a letter, $, or _
  - Can contain letters, digits, underscores, and dollar signs
  - Names are case sensitive
  - Reserved words cannot be used

```
let x = 2;
let y = "Google Earth Engine";
let z = "2";
let x;
x = 2;
```

8

A variable is simply a container for or a reference to data (numbers, strings, dates, images, videos, etc.). By creating variables, you can then work with your data using JavaScript and its associated methods and functions.

This slide highlights some key characteristics of variables specific to JavaScript. First, a variable is declared using the var, let, or const reserved keywords. You can declare a variable and assign data to it in one line of code or declare and assign in separate lines of code. Assignment is done using a single equals sign.

All JavaScript statements should end with a semicolon (similar to a period at the end of a sentence), including variable declarations and assignments. Generally, JavaScript is forgiving if you forget to include semicolons; however, it is considered good and standard practice to include them.

When a variable will store a number, you do not put quotes around the number. When you are assigning text, strings, or numbers that should be treated as strings to a variable, you must include single- or double-quotes.

It is also possible to declare multiple variables at once or on a single line. The declarations are provided in one line and separated with commas.

There are some rules for variable names. Variables must begin with a letter, $, or _. You cannot start a variable with a number. Names are case sensitive (so, x and X would be considered different variables). There are some reserved words that have special uses and meaning in JavaScript (for example, var). So, they cannot be used as variable names.

Lastly, it is standard practice to use camel case for variable names (for example, myMap or myMapObj) where words are merged and the first word is not capitalized. This is not required but is considered a standard amongst JavaScript coders and web developers.

console.log()

```
console.log(x);
console.log(y);
console.log(z);
```

```
» let x = 1;
  console.log(x);

  1
```

9

You will see console.log() used a lot in JavaScript. This is a function that prints something to the console in the web browser. In other words, it allows you to return a result to the console. It is used similar to print() in other commonly used languages.

Scope

- Local Scopes: variables are declared within a function and can only be called in the function
- Global Scope: variables are declared outside of a function and can be called anywhere in the code

10

When working with variables, functions, and scripts, it is important to understand the difference between local and global scope.

When a variable is defined inside of a function, it is not available for use outside of the function. Thus, it has local scope and can only be used or referenced within the function.

In contrast, global scope allows for the variable to be used anywhere in the script and requires that the variable is defined outside of a function.

In short, where a variable is defined in a script can impact how and where it can be used. If you need to use a variable outside of a function, you will need to define it before the function is executed so that it has global scope.

var, let, const

```
var x = 1;
let y = 1;
const z = 1;
```

Declaration Keyword	Scope	Re-Declared/Update	Hoisting
var	Global when declared outside of a function; local when declared inside of a function	Yes/Yes	Hoisted to top of scope and initialized as undefined
let	Global when declared outside of a block; local when declared inside of a block	No/Yes	Hoisted to top of scope but not initialized
const	Global when declared outside of a block; local when declared inside of a block	No/No	Hoisted to top of scope but not initialized

11

Prior to the release of JavaScript ES2015 (ES6), variables were generally declared using the var keyword. However, this is now generally discouraged. Instead, variables should be declared with let or const.

When a variable is declared using the var keyword, it has global scope when declared outside of a function and local scope when declared inside of a function. It can be re-declared and updated. Hoisting means that the declaration of the variable is moved to the top of the scope in which it exists. Variables defined using var are hoisted and initialized with a value of undefined.

In contrast to var, variables declared using let cannot be re-declared; however, they can be updated. They are also hoisted, but they are not initialized as undefined when declared. They have global scope when declared outside of a block and local/block scope when declared inside of a block. A block is code occurring within curly brackets, such as inside of a function or loop.

Variables declared with const are similar to those declared with let except that they cannot be re-declared or updated. These are generally used when the variable is meant to remain constant or not manipulated or updated.

We will use let and const in this course and avoid using var.

Comments

- Comments are meant for humans, and will not be executed by the machine
- Comment your code!!!!

```
//Comment one own line
const x = 4; //Comment after code
/*
Multi-line
comment.*/
```

12

Similar to HTML and CSS, comments can be included with JavaScript in a JavaScript file (.js) or in a <script> tag within an HTML document.

This slide shows how to define single-line and multi-line comments.

It is generally a good idea to comment your code, as this aids in interpretability for later use. Also, lines of code can be "commented out" in the process of debugging or diagnosing errors.

Strict Mode

https://www.w3schools.com/js/js_strict.asp

13

You can run JavaScript code in strict mode by including “use strict” at the top of the script. The goal of this mode is to cause bad syntax to result in errors as opposed to running the poorly formatted code. You can read more about strict mode at the provided link.

Arithmetic Operators

Operator	Use
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation
/	Division
%	Modulus
++	Increment
--	Decrement

14

All coding languages need to be able to perform arithmetic or mathematical operations on numeric data.

This slide shows the defined arithmetic operators for JavaScript specifically.

You may not be familiar with the concept of modulus. This is used to return the remainder after division. For example, 3%2 would yield 1 since there is a remainder of 1 when dividing 3 by 2.

Increment and decrement are used to increase or decreased a value incrementally.

Arithmetic

```
let x = 11;
let y = 7;
let z = x + y;
console.log(z);
```

```
let x = 11;
let y = 7;
let z = x - y;
console.log(z);
```

```
let x = 11;
let y = 7;
let z = x * y;
console.log(z);
```

```
let x = 11;
let y = 7;
let z = x / y;
console.log(z);
```

```
let x = 11;
let y = 7;
let z = x % y;
console.log(z);
```

```
let x = 11;
let y = 7;
let z = x ** y;
console.log(z);
```

```
let x = 5;
x++;
let z = x;
console.log(z);
```

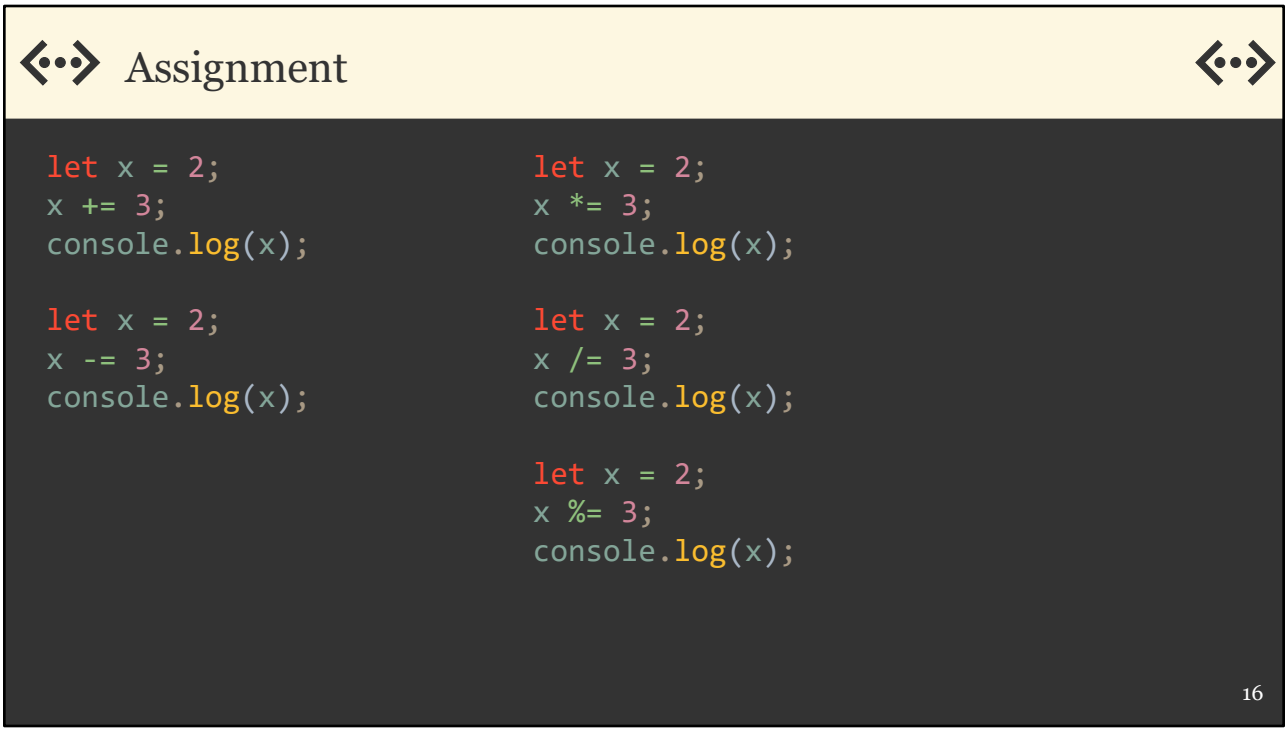
```
let x = 5;
x--;
let z = x;
console.log(z);
```

15

This slide provides some examples of arithmetic operations using JavaScript.

Please work through each of these examples and make sure you understand how and why the result was obtained.

Note here that all variables are declared using the let keyword and each line of code ends with a semicolon.



It is also possible to perform a mathematical operation and automatically assign the result back to the original variable.

In the first column, I am adding 3 to 2 and saving the answer (5) back to the variable x. I then subtract 3 from 2 and save the result (-1) back to the variable x.

In the second column, I am multiplying 2 by 3 and saving the result back to x. Next, I divide by 3. Lastly, I return the modulus.

Comparison Operators

Operator	Use
==	Equal to
===	Equal value and equal type
!=	Not equal to
!==	Not equal value or not equal type
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

17

Comparison operators are used to compare values or values stored in separate variables and include equal to (==), not equal to (!=), greater than (>), greater than or equal to (>=), less than (<), and less than or equal to (<=).

The result will be a Boolean value of true or false depending on whether the comparison is logically true or logically false.

Note that == is used as opposed to = for comparison. This is because = is used for variable assignment, so it cannot also be used for comparison without ambiguity.

It is also possible to test if data have both the same value and the same data type. For example, if one variable stores the value 1 as a number and another variable stores the value"1" as a string, == would yield true while === would yield false, since they have the same value but a different data type.

Logical Operators

Operator	Use
&&	And
||	Or
!	Not

18

To include multiple tests, multiple comparisons can be strung together using logical operators.

AND (&&): Both criteria must be true
OR (||): One of two criteria or both criteria must be true
NOT (!): One criteria but not the other must be true

There is no defined symbol for XOR (A OR B, but NOT A AND B) in JavaScript. However, this logic could be coded if necessary.

Data Types

Data Types

- Numbers

```
let x = 2;
let y = 1.2;
```

- Strings

```
let x = "Remote Sensing";
let y = "1";
```

- Array

```
let a = ["GIS", "Remote Sensing", "Cartography"];
let a = ["text", 1, true, [1,2,3]];
```

20

We will now discuss the data types available in JavaScript.

Numbers store a single number that is treated like a number (mathematical operations can be performed on the value).

Strings store text or numbers that are treated as text (1 vs. “1”).

Both numbers and strings can only store a single feature: one number or one string.

Arrays allow for storing multiple data points as a single variable. Arrays use square brackets, and commas are used to separate each data point. Note that you can store different data types within an array.

Printing Strings

- Print string

```
x = "Geog"
y = 455
z = "Remote Sensing"
console.log("The name of the course is " + x + " " + y + ": " +
z + ".");
```

- Print string using template literal

```
x = "Geog"
y = 455
z = "Remote Sensing"
console.log(`The name of the class is ${x} ${y}: ${z}.`);
```

21

Strings can be combined then logged to the console using the plus sign. Note that spaces within quotes will be included in the generated or printed statement while those outside will not.

Although this works fine, the syntax can be difficult to read and not concise. Another means to print a string is to use a template literal. In a template literal, variables are referenced using ${}. This is similar to the concept of an f-string in Python.

Indexing for Arrays

- Indexing starts at 0 as opposed to 1.

```
let a = ["text", 1, true, [1,2,3]];
let a1 = a[0];
let a2 = a[1];
let a3 = a[2];
let a4 = a[3][0];
console.log(a1);
console.log(a2);
console.log(a3);
console.log(a4);
```

22

That indexing for JavaScript arrays start at 0 as opposed to 1. So, the first element in the array has an index of 0 and the second has an index of 1.

It is common for programming languages to index this way. For example, Python indexing also starts at 0.

Data Types

- Boolean

```
let x = true;
let y = false;
```

- Object

```
let course = {
  code: "Geog",
  number: 456,
  name: "Remote Sensing Applications"
};
```

23

The Boolean data type indicates logical true or logical false. Note that the standard is to use lower case, and you cannot include quotes. If quotes are included, then the data would be treated as a string (true vs. “true). You can also store a set of Boolean values in an array.

An object allows for the storage of different data types within the same variable. For example, an object can store numbers, strings, arrays, and other objects. We will talk about objects in more detail later in the module.

Checking data types

```
let x = 1;
let y = "1";
let z = true;
let a = [1,2,3,4];

let obj = {
  code: "Geog",
  number: 456,
  name: "Remote Sensing Applications",
  courseTitle: function () {
    return this.code + " " + this.number + ":"
+ " " + this.name;
  },
};


function abmult(a, b) {
  return a * b;
}
```

```
console.log(typeof x);
console.log(typeof y);
console.log(typeof z);
console.log(typeof a);
console.log(typeof a[0]);
console.log(typeof obj);
console.log(typeof obj.name);
console.log(typeof abmult);
console.log(typeof obj.courseTitle);
```

24

The typeof keyword can be used to return or determine the data type of a variable.

Data Types

- Types are dynamic for a variable
- Strings can be written in double or single quotes
- Numbers written in quotes will be treated as strings
- If you have quotes as part of the string, use the other type of quote to define the object as a string
- Numbers can be written using scientific notation
- Arrays require square brackets
- Objects are written with curly brackets
- Objects have properties
- Functions are available to convert between data types

25

Here are some general notes on data types in JavaScript.

First, data types are dynamic for a variable. This means that the variable x can store a number then be redefined to store a string.

When defining strings, you must use double- or single-quotes. If a number is written in quotes, it will be treated like a string (mathematical operations can not be performed on it). If you use quotes within a string (for example, sharks with ‘laser” beams attached to their heads), use the other quote type to define the string (for example, let drEvil = “sharks with ‘laser’ beams attached to their heads”).

Numbers can be written using scientific notation (for example, 3.2e3).

Arrays always use square brackets while objects use curly brackets. We will discuss objects in more detail later in the module.

Truthy and Fasly

- Numbers
  - Translate to Boolean true: any non-zero number
  - Translate to Boolean false: 0, NaN, null, undefined
- Strings
  - Translate to Boolean true: any non-empty string
  - Translate to Boolean false: an empty string

26

In JavaScript, truthy values are those that will yield true when converted to a Boolean. For numbers, this includes all numbers except 0. For strings, this includes all non-empty strings. Falsy values are those that yield false when converted to a Boolean. This includes the number 0, empty strings, NaN (not a number), null, and undefined.

Functions

- Function do something when called
- Functions can be assigned a name and they accept parameters
- In order to execute a function, you will provide arguments for each parameter
- Return statements are used to stop the function and have it return something
- Variables declared within a function are local variables

```
function abmult(a, b) {
  return a * b;
}
```

```
let x = abmult(4, 7);
```

27

Functions are designed to do something when called. There are functions built into JavaScript and additional functionality provided by JavaScript libraries. You can also define your own functions.

Functions generally accept user-defined arguments for allowed parameters and return something. A new function can be defined using the function keyword with a list of parameters defined within parentheses. What the function does is defined inside of curly brackets.

In the example on this slide, a function is defined called “abmult” that accepts two arguments. The function will then return the product of the provided numbers. Once the function is defined, it can be used by calling the function name and providing arguments for the parameters.

Note that variables defined within functions can only be used within the function as opposed to anywhere in the script. In other words, they have local scope.

Arrow Functions

```
let abmult = (a, b) => a*b;
let x = abmult(4, 7);
```

28

Arrow functions provide a means to declare simple, one-line functions using a concise syntax. This is similar to lambda functions in Python.

Working with Objects

- Objects allow you to associate many values with a single variable
- Values are written as name:value pairs called properties
- Breaks are not important
- Access properties:
  - course.code
  - course["code"]

```
let course = {
  code: "Geog",
  number: 456,
  name: "Remote Sensing Applications"
};
```

```
console.log(course.code);
console.log(course["code"]);
```

29

We will now discuss JavaScript objects in more detail. First, objects are defined using curly brackets.

Within the object, data are defined as name and value pairs, which are collectively called properties.

To access a specific property, you can call the object name followed by a period and the property name. Alternatively, you can use square brackets as demonstrated on the slide.

In the example, an object is declared called course. This object holds three properties with the names code, number, and name. Two of the variables hold a text string while the other holds a number.

If we wanted to subset out just the course number, this could be accomplished using course.number or course[“number’].

Object notation is used extensively in JavaScript and by web mapping APIs. So, you will get used to seeing a lot of curly brackets in the examples and in your code.

Working with Objects

```
let course = {
  code: "Geog",
  number: 456,
  name: "Remote Sensing Applications",
  courseTitle: function () {
    return this.code + " " +
this.number + ":" + " " + this.name;
  },
};



console.log(course.courseTitle());
```

- Methods are actions that can be performed by objects
- Methods are stored in properties as function definitions
- this is a keyword that means the property from this object
- Access a method:
  - course.courseTitle()
- Call without () will return the function definition

30

Objects can also store functions as a property. Functions stored or associated with an object are generally called methods and define actions that can be performed by or on objects.

In this example, a method called “courseTitle” is defined. The function function is then used to define what the method does. In this case, it returns the entire course title.

To use the method, you need to call it along with the object name: course.courseTitle(). If you would like to return the function definition, then do not include the parenthesis: course.courseTitle.

Methods are meant to be applied only to objects to which they are assigned. They cannot be used or applied to other objects. Again, I think of these as functions tied to a specific object.

Note the use of the keyword this. This indicates a property from the object to which the method is defined. In the example, the properties code, number, and name are being obtained from the current object.

JavaScript Classes

- Blueprint for objects of the same type
- Use constructor method to create a new object of this class
- Pre-defined methods, properties, and options to work with for:
  - Creation
  - Interaction
  - Events
  - Get, set, and change settings and options

31

When using web mapping APIs, you will work with a variety of pre-defined classes that have associated methods to handle creation, interaction, events, and setting and changing options.

Classes provide pre-defined templates for working with and handling types of objects, such as web maps or spatial layers.

Working with Strings

- Backslash (\) escape character turns special characters into string characters

```
let x = 'That\'s mine!';
```

32

This is a note on escape characters. If you must use a special character within a string, this could cause the string to be misinterpreted.

On this slide, a contraction is used, which contains a single quote. This would cause the variable assignment to fail since there are three single quotes.

To alleviate this issue, the backslash escape character can be placed before the special character. This indicates to treat it as text and not a special character, thus alleviating any ambiguity.

Working with Strings

Operation	Use
length	Returns string length
indexOf()	Returns index of first occurrence of specified text
lastIndexOf()	Returns the index of the last occurrence of a specified test in a string
search()	Search for specified value and returns the index
slice()	Returns a portion of a string
substring()	Like slice() but cannot accept negative indexes
substr()	Like slice() but can specify length
replace()	Replaces a specified value with another value
toUpperCase()	Converts to upper case
toLowerCase()	Converts to lower case
concat()	Joins two or more strings
charAt()	Returns character as a specified index
split()	Convert string to an array

https://www.w3schools.com/js/js_string_methods.asp

33

Different data types have methods defined by JavaScript that can be applied to them. This slide provides some examples for string objects.

In the next slide, I will provide some specific examples.

The w3chools.com link included on this slide provides additional examples and descriptions of string methods.

Working with Strings

```
let text1 = "Remote Sensing Applications";
console.log(text1.length);
console.log(text1.replace("Applications", "App"));
console.log(text1.split(" "));
console.log(text1.toUpperCase());
console.log(text1.slice(7, 14));
```

34

Note that methods are applied by calling them after the variable name to which they are meant to be applied. Parentheses must be included, even if no arguments are provided or required.

So, the general form is: variable.method(argument1, argument2).

Work through these examples and make sure you understand how they are applied and why the specific result is obtained.

Working with Numbers

Operation	Use
toString()	Convert number to string
toExponential()	Returns a string using exponential notation with rounding
toFixed()	Returns a string with specified number of decimal places
toPrecision()	Returns a string of a specific length
valueOf()	Returns a number as a number
Number()	Returns a number
parseFloat()	Returns floating point number
parseInt()	Returns integer number

https://www.w3schools.com/js/js_number_methods.asp

35

This slide provide some example methods for number-type variables.

Use the w3schools.com link provided to see additional examples and explanations.

Working with Numbers

```
let num1 = 3;
console.log(typeof num1.toString());
let num2 = 3445892;
console.log(num2.toExponential());
let num3 = 3.56321893;
console.log(num3.toFixed(3));
console.log(typeof num3.toFixed(3));
console.log(typeof Number(num3.toFixed(3)));
```

36

Work through the examples on this slide and make sure you understand why and how the specific results were obtained.

As a note for the toFixed() method, this method is used to round off a number to a defined number of decimal places. However, the number will be converted to a string. So, in the last line, I wrap this process in the Number() function, which converts the string back to a number.

Math Object Methods

Method	Description
abs(x)	Returns the absolute value of x
acos(x)	Returns the arccosine of x, in radians
asin(x)	Returns the arcsine of x, in radians
atan(x)	Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians
atan2(y, x)	Returns the arctangent of the quotient of its arguments
ceil(x)	Returns the value of x rounded up to its nearest integer
cos(x)	Returns the cosine of x (x is in radians)
exp(x)	Returns the value of E^x
floor(x)	Returns the value of x rounded down to its nearest integer

https://www.w3schools.com/js/js_math.asp

37

The Math object is commonly used to perform mathematical operations with JavaScript.

This slide provides some examples of different operations available.

Math Object Methods

Method	Description
log(x)	Returns the natural logarithm (base E) of x
max(x, y, z, ..., n)	Returns the number with the highest value
min(x, y, z, ..., n)	Returns the number with the lowest value
pow(x, y)	Returns the value of x to the power of y
random()	Returns a random number between 0 and 1
round(x)	Returns the value of x rounded to its nearest integer
sin(x)	Returns the sine of x (x is in radians)
sqrt(x)	Returns the square root of x
tan(x)	Returns the tangent of an angle

https://www.w3schools.com/js/js_math.asp

38

This slide provides some additional examples of different Math object methods.

Math Object

```
console.log(Math.PI);
console.log(Math.E);
```

```
console.log(Math.trunc(3.8));
console.log(Math.ceil(3.8));
console.log(Math.floor(3.8));
console.log(Math.round(3.8));
console.log(Math.random(3.8));
console.log(Math.abs(-8.1));
console.log(Math.sqrt(8.1));
```

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

39

Here, I provide some syntax examples that make use of the Math object. Note that the properties (examples in the first column) and functions (examples in the second column) are all prefixed with Math.

Working with Arrays

Operation	Use
join()	Joins all array elements into a string
pop()	Removes the last element from an array
push()	Adds a new element to end of an array
shift()	Removes first array element
unshift()	Adds new element to an array at the beginning
splice()	Add new items to an array at specified index
concat()	Merges arrays into a new array
slice()	Slices out a piece of an array
toString()	Convert array to string
reverse()	Sorts an array alphabetically
Math.max.apply()	Find the highest number in an array
Math.min.apply()	Find the lowest number in an array

https://www.w3schools.com/js/js_array_methods.asp

40

This slide provides example methods for arrays.

The w3schools.com link provides additional examples.

Working with Arrays

```
let arr1 = ["Geography:", "456", "Remote", "Sensing",
"Applications"];

console.log(arr1.join(" "));
arr1.push("Spring", "2019");
console.log(arr1);
arr1.reverse();
console.log(arr1);
let numarr = [1, 3, 5, 8, 9];
console.log(Math.max.apply(null, numarr));
```

41

Work through these examples and make sure that you understand how the results were obtained.

Working with Arrays

Operation	Use
forEach()	Calls a function once for each array element
map()	Creates a new array by performing a function on each array element
filter()	Creates a new array from elements that pass a test
reduce()	Runs a function on each array element to produce a single value
reduceRight()	Like reduce() but works from right-to-left
every()	Check if all array values pass a test
some()	Check if some array values pass a test
indexOf()	Search an array for an element value and returns its position
lastIndexOf()	Like indexOf() but starts are the end of an array
find()	Returns the value of the first array element that passes a test
findIndex()	Like find() but returns the index

https://www.w3schools.com/js/js_array_methods.asp

42

This slide provides additional array methods and examples.

Working with Arrays

```
let num1 = [3, 4, 5, 6, 7, 8];
function addone(value) {
  return value + 1;
}
let num2 = num1.map(addone);
console.log(num2)
```

```
let num1 = [3, 4, 5, 6, 7, 8];
function addgt6(value) {
  return value > 6;
}
let num2 = num1.filter(addgt6);
console.log(num2);
```

```
let num1 = [3, 4, 5, 6, 7, 8];
function addgt6(value) {
  return value > 6;
}
let num2 = num1.every(addgt6);
console.log(num2)
```

43

Here are some additional examples of working with arrays.

The map() method is used to map a function to each element in an array. In the first example, 1 is added to each array element.

The filter() method is used to only return array elements that meet a specific condition defined by a function (in this case, numbers will be returned to the new array variable only if they are larger than 6).

The every() method will return a single Boolean output for the entire array. If all array elements meet the criteria, true will be returned. If at least one array element does not meet the criteria, then false will be returned.

Please work through these examples and make sure you understand them.

Working with Dates

```
let date1 = new Date();
console.log(date1.getFullYear());
console.log(date1.getTime());
console.log(date1.getHours());
console.log(date1.getMonth());

let today = new Date();
let secondDate = new Date();
secondDate.setFullYear(2200, 11, 18);
console.log(today > secondDate);
```

44

Dates can be treated as a unique data type in JavaScript and most other object-oriented languages. So, data defined to the date type will not be treated as numbers or strings, but specifically as dates. Further, there are methods available for working with dates specifically.

On this slide, I am creating a new object called data1 using the Date() function. Note the use of the “new “keyword here. This keyword is generally used when creating a variable using a constructor function, in this case the Date() function.

The Date() function returns the current date and time. For example, I obtained this date when creating this module: Sat Apr 11 2020 17:18:57 GMT-0400 (Eastern Daylight Time).

Components of this date can be extracted using different methods. As demonstrated on this slide, it is possible to extract out the full year, time, hour, and month. Note that getTime() returns the time in milliseconds since the Unix Epoch Time (1-1-1970).

One of the complexities of working with dates is that there are multiple formats available. Thus, standardization can be an issue. If you would like to learn more about working with dates in JavaScript than what is covered here,

please consult w3schools.com, which provides a more in-depth treatment of the topic.

Working with Dates

Operation	Use
getFullYear()	Get year from date (yyyy)
getMonth()	Get month from date (0-11)
getDate()	Get day as number from date (1-31)
getHours()	Get hours from date (0-23)
getMinutes()	Get minutes from date (0-59)
getSeconds()	Get seconds from date (0-59)
getMilliseconds()	Get milliseconds from date (0-999)
getTime()	Get the time (milliseconds since January 1, 1970)
getDay	(Get the weekday as a number (0-6)
Date.now()	Get the time

https://www.w3schools.com/js/js_date_methods.asp

45

This slide provides some example methods applied to Dates. Specifically, these are “get” methods that allow you to extract out or get a portion of the date data.

Working with Dates

Operation	Use
setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

https://www.w3schools.com/js/js_date_methods_set.asp

46

There are also “set’ methods for setting or changing components of the date data.

Working with Booleans

```
let x = 3 > 9;
console.log(x);

let x = 2;
let y = "2";
console.log(x == y);
console.log(x === y);
```

47

This slide provides some examples of operations that will return a Boolean value (true or false) based on whether a condition is logically true or logically false.

Remember that Booleans are their own data type. They are not numbers or strings.

Control Flow, Loops, and Switches

Control Flow

```
let x = 8;
if (x < 10) {
  console.log("Value is less than 10.");
}

let x = 12;
if (x < 10) {
  console.log("Value is less than 10.");
}
x = 8;
if (x >= 10) {
  console.log("Value is greater than or equal to 10.");
} else {
  console.log("Value is smaller than 10.");
}
```

49

Before we move on from this introduction to JavaScript, we need to discuss some common programming language tasks and how they are performed using JavaScript specifically.

We will discuss conditional statements, loops, and switches.

This slide provides examples of conditional statements or control flow. In the first example, a new variable is declared, and the number 8 is assigned to it.

I then set a condition using if. When setting up an if statement, the statement in parentheses (x < 10) must be a test that evaluates to true or false. If the condition evaluates to true, then the code in curly brackets is executed. If it is not true, then the code does not execute, or nothing happens. Since 8 is less than 10, the condition will evaluate to true, and the string will be printed to the console.

If you want something to be executed or returned if the condition evaluates to false, then you can provide an else statement, as in the second example.

Note that the code to execute for “if” and “else” are both written in curly brackets. No condition needs to be defined for the “else” statement since this is the default code to execute if the “if” condition evaluates to false.

Control Flow

```
let x = 8;
if (x === 10) {
  console.log("Value is 10.");
} else if (x < 10) {
  console.log("Value is less than 10.");
} else {
  console.log("Value is larger than 10.");
}
```

50

What if you want to have more than two potential outcomes?

This can be accomplished by including one or multiple else if statements.

The ”if” and all “else if” statements must include conditions. Again, the “else” statement is the default so does not require a condition.

In the example, if the value is 10, the output defined by the if statement will be executed. If the condition defined by if else evaluates to true, then that statement will be executed (which would be the case in this example). If none of the conditions evaluate to true, or the value is larger than 10, code from the else statement will be executed.

Note that you can have multiple “else if” statements.

Switch

```
let x = "Four Stars";
let t;
switch (x) {
  case "One Star":
    t = "This is a One Star Restaurant.";
    break;
  case "Two Stars":
    t = "This is a Two Star Restaurant.";
    break;
  case "Three Stars":
    t = "This is a Three Star Restaurant.";
    break;
  case "Four Stars":
    t = "This is a Four Star Restaurant.";
    break;
  case "Five Stars":
    t = "This is a Five Star Restaurant.";
    break;
  default:
    t = "Restaurant was not rated.";
}
console.log(t);
```

51

If you have a lot of conditions to test, using if, if else, and else statements can be cumbersome. As an alternative, you can use a switch. In a switch, each condition is specified using the case keyword, code to execute or text to print is defined for each case, and each case ends with break. It is also common to include a default if none of the cases evaluate to true. Note the use of curly brackets.

In short, if you are testing multiple conditions, a switch will likely require less code and be more interpretable than the conditional statement that would yield the same result.

For Loop

```
for (i = 0; i < 10; i++) {
  console.log("The value is " + i + ".");
}

for (i = 0; i < 10; i++) {
  console.log(`The value is ${i}`);
}
```

52

What if you would like to perform a task repeatedly for a set of data values? For example, what if you want to multiply all the numbers in an array by the same value or read in and process all the files from a folder?

This can be accomplished with loops. We will begin with a discussion of for loops. A for loop is designed to execute a task for all elements in a set of elements. The for loop can accept three statements. The first statement is executed before entering the loop. The second defines the condition for executing the loop, and the last statement specifies what to do once each iteration has been executed.

In the provided example, the variable i is set to 0 initially. Next, “The value is 0.” is printed to the console. Next, 1 is added to 0 (to get 1), the loop is executed again, and a new print statement is logged (“The value is 1.”). This process continues until the condition (the second statement) is not met. So, it will execute until x reaches 10. At that point, the loop is complete.

For Loop

```
let countries = ["Belgium", "Germany", "Iceland",
"Hungary", "Scotland"];

let i = 0;
let len = countries.length;
for (; i < len; i++) {
  console.log("I would like to visit" + " " +
countries[i] + ".");
}
```

53

This is another example of a “for loop”. First, I am creating a variable called countries that stores a set of country names as strings in an array. I then define a variable i and assign it the number 0. A third variable (len) is assigned the number of elements in the countries array.

The for loop is then used to loop through all the countries in the array. Note that no initial condition is defined or necessary, and “i” is used to specify the index of each country in the array.

Why is the loop set to run until i is less than the length of the array as opposed to less than or equal to the length of the array? This is because JavaScript starts indexing at 0 as opposed to 1. So, the last element in the array has an index of n-1 as opposed to n, where n is the number of elements in the array.

Note that for loops can be combined with control flow to execute different sets of code depending on a set of conditions.

For Loop + Flow Control

```
for (i = 0; i < 15; i++) {
  if (i == 10) {
    console.log("Value is 10.");
  } else if (i < 10) {
    console.log("Value is less than 10.");
  } else {
    console.log("Value is larger than 10.");
  }
}
```

54

This code demonstrates combining a for loop and control flow to allow for different results for each iteration of the loop.

For Loop + Flow Control

```
a = [1,2,3,4,5,6,7,8,9];
let i = 0;
aLen = a.length;
for (; i < aLen; i++) {
   if (a[i]%2 === 0){
      console.log(`${a[i]} is even.`);
   } else {
      console.log(`${a[i]} is odd.`);
   }
}
```

55

This example combines a for loop and flow control to print whether a value is positive or negative. The tests make use of modulus. Note the use of template literals in the print statements.

continue

```
a = [1,2,3,4,5,6,7,8,9];
let i = 0;
aLen = a.length;
for (; i < aLen; i++) {
   if (a[i]%2 === 0){
      console.log(`${a[i]} is even.`);
   } else {
      continue;
   }
}
```

56

Continue is used to skip an iteration of a loop. In the example, odd values are skipped.

break

```
a = [1,2,3,4,5,6,7,8,9];
let i = 0;
aLen = a.length;
for (; i < aLen; i++) {
   if (a[i]%2 === 0){
    break;
   } else {
    console.log(`${a[i]} is odd.`);
   }
}
```

57

In contrast to continue, break causes the loop to stop. In the example, the loop stops when the first even number is reached.

While Loop

```
let i = 20;
while (i > -1) {
  console.log("The number is " + i);
  i--;
}

let i = 0;
while (i < 20) {
  console.log("The number is " + i);
  i++;
}
```

Watch for infinite loops!!!!

58

Another loop option is the while loop. A while loop will continue to execute until the condition is not met.

For example, this loop will execute until i reaches 20. If we initiate i at 21, and 1 is added each time using the increment defined in the loop (i++), then the condition will always be met. So, the loop will never stop. This is known as an infinite loop and is an issue if while loops are not set up correctly. The loop will continue until it is manually terminated or the computer runs out of memory.

While Loop

```
let countries = ["Belgium", "Germany", "Iceland",
"Hungary", "Scotland"];
let i = 0;

while (countries[i]) {
  console.log("I would like to visit" + " " +
countries[i] + ".");
  i++;
}
```

59

This is another example of a while loop. In this case, the loop will continue until it reaches an index (i) that is not in the dataset.

Events

- Provides funtionality to webpages
- Allow webpages to accept input and respond or change in some way

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

```
<button onclick="this.innerHTML = Date()">The time is?</button>
```

60

One of the primary uses of JavaScript is to add functionality to a webpage or define what should happen as a result of a certain event.

For example, what should happen when a button is clicked or a form is filled out?

In the context of web mapping, it is also used to define the functionality of the map and map elements.

We will discuss event handling and manipulation of the document object model (DOM) using JavaScript in a later module.

GeoJSON

GeoJSON

- JSON = JavaScript Object Notion
- Used to store and exchange data
- Text written with JavaScript object notation
- GeoJSON = allows you to encode geographic structures
- Used in web applications
- One file can store multiple geometry types

http://geojson.org/

62

JSON stands for JavaScript Object Notation, which was discussed earlier in the module. JavaScript is capable of interacting with data provided in this format.

Since JavaScript is the primary client-side web programming language, JSON is often used to represent data on the web or transfer data between the client and server.

GeoJSON expands JSON to include geographic features and spatial reference information. This is one means to work with spatial data on the web.

A JSON file can store multiple geometry types (points, lines, and polygons) in the same file.

The next set of slides provide some examples.

GeoJSON: Point


```
{
"type": "FeatureCollection",
"name": "json_point2",
"crs": { "type": "name", "properties": {
"name": "urn:ogc:def:crs:OGC:1.3:CRS84" }
},
"features": [
{ "type": "Feature", "properties": { "id": 1,
"attribute1": 1, "attribute2": "point1",
"attribute3": 0.1111 }, "geometry": { "type":
"Point", "coordinates": [ -96.437, 37.283 ] } }
]
}
```


63

The text on this page represents the GeoJSON code for the point shown on the map.

It contains a feature type, a file name, a coordinate reference system, or crs, definition, a list of attribute names and values, and the x and y coordinate representing the point.

This structure makes use of objects and arrays to define this geographic feature as text or code.

GeoJSON: Point


```
{
"type": "FeatureCollection",
"name": "example_point",
"crs": { "type": "name", "properties": { "name":
"urn:ogc:def:crs:OGC:1.3:CRS84" } },
"features": [
{ "type": "Feature", "properties": { "id": 1, "attribute1": 1,
"attribute2": "point1", "attribute3": 0.1111 }, "geometry": { "type":
"Point", "coordinates": [ -96.437, 37.283 ] } },
{ "type": "Feature", "properties": { "id": 2, "attribute1": 2,
"attribute2": "point2", "attribute3": 0.22222 }, "geometry": {
"type": "Point", "coordinates": [ -83.006, 39.913 ] } },
{ "type": "Feature", "properties": { "id": 3, "attribute1": 3,
"attribute2": "point3", "attribute3": 0.33333 }, "geometry": {
"type": "Point", "coordinates": [ -115.666, 43.247 ] } },
{ "type": "Feature", "properties": { "id": null, "attribute1": null,
"attribute2": null, "attribute3": null }, "geometry": { "type":
"Point", "coordinates": [ -78.691, 40.501 ] } }
]
}
```


64

This example is similar to the previous one except that this file now contains four points.

For each point, attributes and x and y coordinates are provided.

GeoJSON: Line


```
{
"type": "FeatureCollection",
"name": "line_example",
"crs": { "type": "name", "properties": { "name": "urn:ogc:def:crs:OGC:1.3:CRS84" } },
"features": [
{ "type": "Feature", "properties": { "id": 1, "attribute1": 1, "attribute2": 1111, "attribute3": "line" }, "geometry": {
"type": "MultiLineString", "coordinates": [ [ [ -118.903, 39.471 ], [ -112.086, 41.188 ], [ -105.957, 42.315 ], [ -
97.718, 41.237], [ -90.313, 39.030], [ -85.458, 36.234], [ -78.544, 38.883 ] ] ] } }
]
}
```


65

This example represents GeoJSON code for a line.

This is pretty similar to the code for point features, except that the line is now represented as a series of x and y coordinate pairs as opposed to a single coordinate.

The attribute definition is the same as that for the point.

GeoJSON: Polygon


```
{
"type": "FeatureCollection",
"name": "polygon_example3",
"crs": { "type": "name", "properties": { "name":
"urn:ogc:def:crs:OGC:1.3:CRS84" } },
"features": [
{ "type": "Feature", "properties": { "id": 1 },
"geometry": { "type": "MultiPolygon",
"coordinates": [ [ [ [ -101.2, 43.051 ], [ -99.238,
45.111 ], [ -99.238, 45.111 ], [ -93.697, 46.239 ],
[ -89.332, 42.463 ], [ -88.254, 36.333 ], [ -
92.275, 31.429 ], [ -100.219, 32.312 ], [ -
107.183, 36.235 ], [ -105.809, 42.07 ], [ -101.2,
43.051 ] ], [ [ -99.581, 40.354 ], [ -98.503,
34.371 ], [ -93.501, 36.823 ], [ -92.128, 39.373
], [ -95.364, 41.923 ], [ -99.581, 40.354 ] ] ] ] }
}
]
}
```


66

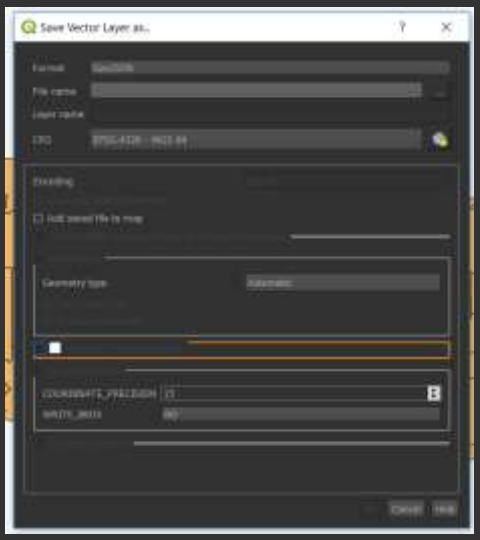
This final example is the result for a polygon feature.

Similar to a line feature, this polygon is defined based on a set of coordinate pairs. The first and last coordinate are identical so that the feature is closed.

This specific feature has an internal ring representing a hole. This hole is defined by coordinate pairs also.

GeoJSON in QGIS

- Can export files to GeoJSON format
- Can create bounding box
- Can set coordinate precision
- QGIS can read and write many file types



67

GeoJSON files are directly readable in the free and open-source QGIS software.

Data can be read in, and other data types can be saved to GeoJSON.

You can specify a bounding box as an extent and also the coordinate precision. Reducing the number of decimal values stored will decrease the file size.

One benefit of QGIS is that it can read in and convert a wide variety of data types.

I often use QGIS as a data conversion tool for this reason.

This is the end of this lecture module.

Please return to the West Virginia View Webpage for additional content.

Thanks! Hope you found this useful.