# CS355 - Topic 1: JavaScript Basics

## **Dates:** January 26, 2026 and January 28, 2026

### January 26:

#### JavaScript Overview:

- JavaScript is:
  - Multi-paradigm
  - Procedural
  - Functional
  - Event-driven
  - Object-oriented
  - Weakly typed
- JavaScript can be considered more object-oriented than Java
- Creating an object in JavaScript (without having to create a class):

```
let myfraction = {
    num: 3,
    den: 4,
    toDecimal: function() { return this.num / this.den }
};
```

#### Compiled vs Interpreted Languages

- Compiled Languages (C, C++, Rust, ...)
  - Source code -> Compile -> Machine Code -> Executable -> Run -> Output
- Intepretted Languages (Python, PHP, ...)
  - Source Code -> Interpreter -> Output
- Compiled languages tend to be faster because:
  - Compiler optimizations such as (loop unraveling, dead code elimination, ...)
  - The compilation is slow because of these optimization BUT because of it, the code can RUN fast
- JavaScript is both compiled AND interpreted
  - The JavaScript engine (v8) has a JavaScript interpreter (Ignition) and a JavaScript compiler (Turbofan)

---

### January 28:

#### Printing:

```
console.log()
```

- Whatever is passed into the function will be printed to the console

```
console.table()
```

- Formats 2D arrays into a table, each 1D array is a row

### Primitive Data Types:

- JavaScript has 7 primitives:
    1. number
    2. string
    3. boolean
    4. undefined
    5. null
    6. symbol (not discussed in class)
    7. BigInt (not discussed in class)
- `typeof` returns the data type of any data
- All types of numbers fall under one umbrella which is IEEE 754 64-bit doubles
- Strings can be inside single quotes or double quotes, an occasionally you will see strings inside of backticks when they contain expressions

```
let message = "Hello world";
let messageTwo = 'Hello world';
let name = "Alfred";
let messageThree = `Hello, ${name}`;
```

- Undefined vs Null:
    - Undefined is the value a variable has when it is declared but not initialized
    - Null is an explicit declaration that this value does not exist / make sense

    ```
    let John = { employeeID: 1234, name: "John", supervisor: "Jane"};
    let Marko = { employeeID: 2345, name: "Marko", supervisor: undefined
    };
    let Jane = { employeeID: 1, name: "Jane", supervisor: null };
    ```

    - Marko's supervisor is undefined because he is a new hire and does not have a supervisor **yet**
    - Jane's supervisor is null because she own the company and does not and never will have a supervisor
- Anything that is not primitive is an Object
- Primitives are **not** decomposible meaning we can't go into them using the dot operator
- Numbers, strings, and booleans have object wrappers which allows us to perform some dot operators on them. For instance, `"hello world".length` should not work because `"hello world"` is a string **BUT** JavaScript engine understands what you are trying to do when interpreting

your code and uses syntactic sugar to re-write your code using the string object wrapper to make it work as intended

## Objects:

- Objects are defined inside of curly braces as key value pairs

```
let csci355 = { name: "Internet and Web Technologies", teacher: "Raymond Law", prereq: "csci313" };
```

- Once an object is defined, its properties can be access with the dot operator and its values can be changed

```
let player = { name: "LeBron James", jerseyNumber: 6, currentTeam: "Los Angeles Lakers" };
player.currentTeam = "New York Knicks";
```

- When you declare variables with the keyword `let`, these variables are mutuable meaning they can be changed
- When you delcare variables with the keyword `const`, these variables are immutable meaning they cannot be changed after initialization AND it requires to be initialized with a value when declared (can't declare with no value)
- Deconstruction Assignment:
  - Say you import a large JSON object but you only want a specific few pieces of data from that object then you would utilize the deconstruction assignment

  ```
  const downloadedData = { a: 1, b: 2, c: 3, d: 4, e: 5, ...};
  const { c, d, e } = downloadedData;
  ```

  - This gives us free access to use variables c, d, and e from the downloadedData object and nothing else from the object. You could also do some reassignments if you don't like the original variable names

  ```
  const { c: cat, d: dog, e: elephant } = downloadedData;
  ```

- Object shorthand:

```
let a = 10;
let b = 20;
let c = 30;

let myObject = { a, b, c }
```

## Functions:

- Functions in JavaScript are first class, meaning it is treated like any other variable would be

```
let sq = function(x) { return x * x };
```

- Higher Order Function:
  1. Takes another function as input
  2. Returns another function as output
  3. Both 1 and 2
- Function declaration:

```
function sum(a,b) {
  return a + b;
};
```

- All function declarations are identified by the JavaScript engine when interpreted and moved to the top of the code so that the function can be used on any line (use the function on a line prior to the the line containing the function declaration)
- Function Expression:

```
const sum = function(a,b) {
  return a + b;
};
```

- Arrow Functions:

```
const sum = (a,b) => a + b;
```

- Arrow function tend to be a little bit more tricky than function expression. They are clean for one liners but tend to get more complicated with multi-liners (example later)
- The function sum has two parameters so they are nested inside of parentheses, for one parameter the parentheses are not needed but including them doesn't hurt, and for a function with zero parameters any empty set of parentheses is **required**

```
const sumOfSquare = (a,b) => {
  let aa = a * a;
  let bb = b * b;
  return aa + bb;
}
```

- Example of a multi-line function with with arrow function

## Spread and Rest Operators:

- Spread: Glue things together for arrays and objects

```
let a = [1,2,3];
let b = [5,6,7];
let c = [...a, 4 , ...b];  // c = [1,2,3,4,5,6,7]

let downloadedData = { b: 2, c: 3, d: 4, e: 5};
let a = 1;
let desiredData = { a, ...downloadedData };
```

- Rest: Used inside of function parametes when the number of parameters cannot be fixed

```
function maxNumberInList(...numbers);
```

- If you were trying to make your own implementation of the max function you would need to utilize the rest operator because the built in max function works with any number of parameters passed in