SECTION 2: JavaScript Fundamentals - Part 1

Lecture 2: Hello World!

In this lecture, we write our very first JavaScript code using the browser's console. The console is a great place for testing code and fixing errors, though we won't use it for real applications later.

We can open the console in Chrome using:

- Command + Option + J (Mac) / Ctrl + Alt + J (Windows)
- Right-click + Inspect → Console tab
- Menu → View → Developer → JavaScript Console

Let's write our first JavaScript code:

```
alert('Hello World!');

// We can write any JavaScript and it gets immediately evaluated
const javascript = 'amazing';
if (javascript === 'amazing') alert("Let's learn JavaScript!");

// We can also do math
40 + 8 + 23 - 10; // 61

// Store results in variables
const totalHours = 40 + 8 + 23 - 10;
totalHours; // 61
```

The alert creates a popup with our text. This is just a taste of what JavaScript can do!

Lecture 3: A Brief Introduction to JavaScript

JavaScript is a high-level, object-oriented, multi-paradigm programming language. But what does that actually mean?

As a programming language, JavaScript allows us to write code that instructs computers to perform actions. Being high-level means we don't need to worry about complex details like memory management - there are abstractions that make the language easier to write and learn.

JavaScript is object-oriented, meaning it's based on the concept of objects for storing data. It's also multi-paradigm, allowing different programming styles like imperative and declarative programming.

JavaScript was created in 1995 by Brendan Eich while working at Netscape, originally named Mocha, then LiveScript, and finally JavaScript (as a marketing decision to piggyback on Java's popularity, though the languages aren't related).

JavaScript has evolved significantly over time:

- ES5 (2009) was the standard for many years
- ES6/ES2015 (2015) was a major update with many new features
- Since then, we get new releases every year (ES2016, ES2017...)

JavaScript runs in browsers through the JavaScript engine built into each browser (V8 in Chrome, SpiderMonkey in Firefox). Today, JavaScript can also run outside browsers using Node.js, allowing for backend development, meaning we can build full-stack applications with JavaScript alone.

Lecture 4: Linking a JavaScript File

While we previously wrote code directly in the console, real development involves writing code in files. Let's properly connect JavaScript to our HTML document.

We have two options:

1. Write JavaScript directly in HTML using inline script tags:

```
<script>
  const javascript = 'amazing';
  if (javascript === 'amazing') alert("Let's learn JavaScript!");
  console.log(40 + 8 + 23 - 10);
</script>
```

2. Link to an external JavaScript file (the better approach):

```
<!-- In HTML file -->
<script src="script.js"></script>
```

```
// In script.js file
const javascript = 'amazing';
if (javascript === 'amazing') alert("Let's learn JavaScript!");
console.log(40 + 8 + 23 - 10);
```

When using external files, we separate our website content (HTML) from the JavaScript logic, making code more maintainable. This approach is better for larger projects.

Note that we need to use `console.log()` to see results in the console when writing code in a file, unlike when we're working directly in the console.

Lecture 5: Values and Variables

Values are the most fundamental unit of information in programming - pieces of data we want to use or store. We can store these values in variables to reuse them.

```
// Values
console.log('Jonas'); // String value
console.log(23); // Number value
console.log(23 + 7); // Expression creating a new value (30)

// Storing values in variables
let firstName = 'Jonas';
console.log(firstName);
console.log(firstName);
```

When naming variables, follow these rules:

- Use camelCase (first word lowercase, subsequent words capitalized)
- Names can contain letters, numbers, underscores, or $ signs
- Cannot start with a number
- Cannot use reserved JavaScript keywords
- Variable names should be descriptive

```
// Valid variable names
let _age = 26;
let $name = 'Jonas';
let myFirstJob = 'Programmer'; // Descriptive - good!

// Invalid variable names
let 3years = 3; // Cannot start with number
let jonas&matilda = 'JM'; // No special characters
let new = 27; // Cannot use reserved keyword

// Less descriptive - avoid
let job1 = 'Programmer';
let job2 = 'Teacher';
```

Lecture 6: Data Types

In JavaScript, values can have different types. There are two main categories:

1. Primitive types: Number, String, Boolean, Undefined, Null, Symbol, BigInt
2. Object types: Objects, Arrays, Functions, etc.

JavaScript has dynamic typing - we don't manually define types. The type is automatically determined when a value is stored and can change when the variable is assigned a new value.

```
// Different data types
let javascriptIsFun = true; // Boolean
console.log(typeof javascriptIsFun); // "boolean"
console.log(typeof 23); // "number"
console.log(typeof 'Jonas'); // "string"

// Dynamic typing in action - type can change
javascriptIsFun = 'YES!';
console.log(typeof javascriptIsFun); // Now "string"

// Undefined type
let year; // Value is undefined
console.log(year); // undefined
console.log(typeof year); // "undefined"

// Later define the variable
year = 1991;
console.log(typeof year); // Now "number"

// Note: typeof null returns "object" - this is a bug!
console.log(typeof null); // "object" (should be "null")
```

Lecture 7: Let, Const, and Var

JavaScript has three ways to declare variables:

```
// 1. let - for variables that can change (mutable)
let age = 30;
age = 31; // Reassigning is allowed

let firstName; // Can declare empty variables with let

// 2. const - for variables that shouldn't change (immutable)
const birthYear = 1991;
// birthYear = 1990; // Error! Cannot reassign
// const job; // Error! Must be initialized

// 3. var - old way of defining variables (avoid using)
var job = 'Programmer';
job = 'Teacher'; // Can be reassigned like let
```

As a best practice, use `const` by default and only use `let` when you're sure the variable needs to change. This minimizes variable mutations and potential bugs. Avoid `var` completely as it has function scope rather than block scope (we'll learn more about this later).

You can actually create variables without any keyword, but this is terrible practice as it creates properties on the global object instead of properly scoped variables:

```
// Don't do this!
lastName = 'Schmedtmann'; // Creates a global object property
```

Lecture 8: Basic Operators

Operators allow us to transform values and combine multiple values.

```
// Math operators
const now = 2037;
const ageJonas = now - 1991; // 46
const ageSarah = now - 2018; // 19
console.log(ageJonas, ageSarah);

console.log(ageJonas * 2); // 92
console.log(ageJonas / 10); // 4.6
console.log(2 ** 3); // 2³ = 8 (Exponentiation)

// Concatenation with + operator
const firstName = 'Jonas';
const lastName = 'Schmedtmann';
console.log(firstName + ' ' + lastName); // "Jonas Schmedtmann"

// Assignment operators
let x = 10 + 5; // 15
x += 10; // x = x + 10 = 25
x *= 4; // x = x * 4 = 100
x++; // x = x + 1 = 101
console.log(x);

// Comparison operators (return boolean values)
console.log(ageJonas > ageSarah); // true
console.log(ageSarah >= 18); // true
console.log(now - 1991 > now - 2018); // true
```

Lecture 9: Operator Precedence

JavaScript has a well-defined order in which operators are executed, just like in mathematics.

```
const now = 2037;
const ageJonas = now - 1991;
const ageSarah = now - 2018;

// Math before comparison
console.log(now - 1991 > now - 2018); // 46 > 19, so true

// Left-to-right execution for same-precedence operators
console.log(25 - 10 - 5); // (25 - 10) - 5 = 10

// Right-to-left for assignment
let x, y;
x = y = 25 - 10 - 5; // x = y = 10, so x = 10, y = 10
console.log(x, y);

// Grouping with parentheses (highest precedence)
const averageAge = (ageJonas + ageSarah) / 2;
console.log(averageAge); // (46 + 19) / 2 = 32.5
```

The complete precedence table can be found in the MDN documentation, but here's a simplified order:

1. Parentheses `()`
2. Increment/decrement `++/--`
3. Exponentiation `**`
4. Multiply/divide `*/`
5. Add/subtract `+/-`
6. Comparison `< > <= >=`
7. Equality `== !=`
8. Assignment `= += -= *= /=`

Lecture 11: Strings and Template Literals

Template literals make it much easier to create strings with variables and expressions embedded in them.

```
const firstName = 'Jonas';
const job = 'teacher';
const birthYear = 1991;
const year = 2037;

// Old way (string concatenation)
const jonasOld =
  "I'm " + firstName + ', a ' + (year - birthYear) + ' year old ' +
job + '!';
console.log(jonasOld);

// New way (template literals)
const jonasNew = `I'm ${firstName}, a ${year - birthYear} year old
${job}!`;
console.log(jonasNew);

// Template literals also work for regular strings
console.log(`Just a regular string!`);

// Multi-line strings with template literals
console.log(`String with
multiple
lines`);

// Old way of multi-line strings (less clean)
console.log(
  'String with \n\
multiple \n\
lines',
);
```

Template literals (backticks) are much more readable and easier to write, making them one of the most useful ES6 features.

Lecture 12: Taking Decisions: if/else Statements

Conditional statements let us execute different code based on different conditions, giving us more control over our program flow.

```
const age = 19;

// If/else conditional statement
if (age >= 18) {
  console.log('Sarah can start driving license 🚗 ');
} else {
  const yearsLeft = 18 - age;
  console.log(`Sarah is too young. Wait another ${yearsLeft} years ⏳
`);
}

// Creating variables conditionally
const birthYear = 1991;
let century;

if (birthYear <= 2000) {
  century = 20;
} else {
  century = 21;
}
console.log(century);
```

The if/else statement structure is:

```
if (condition) {
  // Code executed when condition is TRUE
} else {
  // Code executed when condition is FALSE (optional)
}
```

Lecture 14: Type Conversion and Coercion

JavaScript handles types in two ways: conversion (explicit) and coercion (automatic).

```
// Type Conversion (explicit)
const inputYear = '1991';
console.log(Number(inputYear) + 18); // 2009

// Conversion examples
console.log(Number('Jonas')); // NaN (Not a Number)
console.log(String(23)); // "23"

// Type Coercion (automatic)
console.log('I am ' + 23 + ' years old'); // String concatenation
// The + operator triggers coercion to string

console.log('23' - '10' - 3); // 10
console.log('23' * '2'); // 46
console.log('23' > '18'); // true
// Other operators trigger coercion to number

// Mixed coercion example
let n = '1' + 1; // '11' (string)
n = n - 1; // 10 (number)
console.log(n);
```

Key points:

- The + operator converts numbers to strings when used with a string
- The -, *, /, >, < operators convert strings to numbers
- Understanding coercion helps you write cleaner code and avoid unexpected bugs

Lecture 15: Truthy and Falsy Values

Falsy values are values that aren't exactly `false` but become `false` when converted to a boolean. There are only 5 falsy values in JavaScript:

- `0`
- `''` (empty string)
- `undefined`
- `null`
- `NaN`

Everything else is truthy - they convert to `true`.

```
console.log(Boolean(0)); // false
console.log(Boolean('')); // false
console.log(Boolean(undefined)); // false
console.log(Boolean('Jonas')); // true
console.log(Boolean({})); // true

// In practice, conversion to boolean happens implicitly
// For example, in conditions:
const money = 0;

if (money) {
  console.log("Don't spend it all!");
} else {
  console.log('You should get a job!'); // This executes because 0 is
falsy
}

// Be careful with 0
let height;
if (height) {
  console.log('Height is defined');
} else {
  console.log('Height is UNDEFINED'); // This executes because height
is undefined
}
```

Lecture 16: Equality Operators: == vs. ===

JavaScript has two equality operators with different behaviors:

```
const age = 18;

// Strict equality (===) - NO type coercion
console.log(age === 18); // true
console.log('18' === 18); // false

// Loose equality (==) - WITH type coercion
console.log(age == 18); // true
console.log('18' == 18); // true

// Practical example
const favourite = Number(prompt("What's your favourite number?"));

if (favourite === 23) {
  console.log('Cool! 23 is an amazing number!');
} else if (favourite === 7) {
  console.log('7 is also a cool number');
} else {
  console.log('Number is not 23 or 7');
}

// Inequality operators
const notEqual = favourite !== 23;
console.log(notEqual);
```

As a best practice, always use strict equality (`===`) to avoid unexpected type coercion bugs. If you need to compare values of different types, convert them explicitly before comparison.

Lecture 17: Boolean Logic

Boolean logic uses TRUE and FALSE values combined with logical operators:

- AND (&&): TRUE only if ALL operands are TRUE
- OR (||): TRUE if at least ONE operand is TRUE
- NOT (!): Inverts the value (TRUE becomes FALSE, FALSE becomes TRUE)

Truth tables and examples:

- A AND B: TRUE only when both A and B are TRUE
- A OR B: TRUE when either A or B or both are TRUE
- NOT A: TRUE when A is FALSE, FALSE when A is TRUE

We'll apply these in the next lecture with JavaScript logical operators.

Lecture 18: Logical Operators

JavaScript implements boolean logic with the && (AND), || (OR), and ! (NOT) operators.

```
const hasDriversLicense = true; // A
const hasGoodVision = true; // B
const isTired = false; // C

// AND operator
console.log(hasDriversLicense && hasGoodVision); // true

// OR operator
console.log(hasDriversLicense || hasGoodVision); // true

// NOT operator
console.log(!hasDriversLicense); // false

// Practical example
const shouldDrive = hasDriversLicense && hasGoodVision && !isTired;

if (shouldDrive) {
  console.log('Sarah is able to drive!');
} else {
  console.log('Someone else should drive...');
}

// Complex conditions
console.log(hasDriversLicense && hasGoodVision && !isTired); // true
```

These operators are essential for creating complex conditions in if/else statements.

Lecture 20: The Switch Statement

The switch statement is an alternative to if/else when comparing a value against multiple options.

```
const day = 'monday';

switch (day) {
  case 'monday': // day === 'monday'
    console.log('Plan course structure');
    console.log('Go to coding meetup');
    break; // Without break, code would continue to next case
  case 'tuesday':
    console.log('Prepare theory videos');
    break;
  case 'wednesday':
  case 'thursday': // Same code for both days
    console.log('Write code examples');
    break;
  case 'friday':
    console.log('Record videos');
    break;
  case 'saturday':
  case 'sunday':
    console.log('Enjoy the weekend :D');
    break;
  default: // Like 'else' - runs if no case matches
    console.log('Not a valid day!');
}

// Equivalent if/else structure
if (day === 'monday') {
  console.log('Plan course structure');
  console.log('Go to coding meetup');
} else if (day === 'tuesday') {
  console.log('Prepare theory videos');
} else if (day === 'wednesday' || day === 'thursday') {
  console.log('Write code examples');
} else if (day === 'friday') {
  console.log('Record videos');
} else if (day === 'saturday' || day === 'sunday') {
  console.log('Enjoy the weekend :D');
} else {
  console.log('Not a valid day!');
}
```

The switch statement can be more readable for cases where you're comparing a single value against multiple options.

Lecture 21: Statements and Expressions

Understanding the difference between statements and expressions helps write better JavaScript.

- **Expression**: A piece of code that produces a value

```
3 + 4;
1991;
true && false && !false;
```

- **Statement**: A larger piece of code that doesn't produce a value by itself

```
if (23 > 10) {
  const str = '23 is bigger!';
}
```

In template literals, we can only insert expressions, not statements:

```
console.log(`I'm ${2037 - 1991} years old`); // Works (expression)
// console.log(`I'm ${if (age > 18) { 'an adult' }}`); // ERROR (statement)
```

Lecture 22: The Conditional (Ternary) Operator

The ternary operator allows us to write compact if/else statements in a single line.

```
const age = 23;

// Ternary operator
const drink = age >= 18 ? 'wine 🍷' : 'water 💧';
console.log(drink); // "wine 🍷"

// Equivalent if/else
let drink2;
if (age >= 18) {
  drink2 = 'wine 🍷';
} else {
  drink2 = 'water 💧';
}
console.log(drink2); // "wine 🍷"

// Using ternary in template literal (since it's an expression)
console.log(`I like to drink ${age >= 18 ? 'wine 🍷' : 'water 💧'}`);
```

The ternary operator is perfect for quick decisions, especially when we need to use the result as a value in another operation. However, it doesn't replace if/else for complex logic with multiple lines of code.

Lecture 24: JavaScript Releases: ES5, ES6+, and ESNext

JavaScript has evolved significantly since its creation:

- **ES5 (2009)**: The standard for many years
- **ES6/ES2015 (2015)**: A major update with many features we've used:
  - let/const
  - Arrow functions
  - Template literals
  - Classes
  - Destructuring
  - And many more!
- **ES2016-ES2022**: Annual smaller releases with new features

Modern browsers automatically update, supporting the latest features. For older browsers, we can use tools like Babel to convert modern code to ES5.

The language continues to evolve, with proposals for new features constantly being considered through a 4-stage process before becoming part of the official standard.