Intro to JavaScript

An introduction to JavaScript including a brief overview of its history, how it's used in the industry, and how to write and run JavaScript programs.

JavaScript and Programming Languages

What is JavaScript?

JavaScript was originally created to make things happen on websites.

Any interactivity or dynamic functionality of a website is usually done using JavaScript.

Making websites interactive is still one of the main purposes of JavaScript today.

But it's also used for several other things, such as handling back-end logic on servers.

About programming languages

A programming language is a type of ***formal language***.

- Formal languages aren't like ***natural languages*** that people speak (ex.: English)
- Formal languages are *created and designed by people* for specific purposes.
  - Ex.: programming, chemical notation (H_2O), arithmetic ($sin(\theta \pm n * \Delta)$)

Natural languages can break rules and still have meaning…

> The great fall of the offwall entailed at such short notice the pftjschute of Finnegan, erse solid man, that the humptyhillhead of humself prumptly sends an unquiring one well to the west in quest of his tumptytumtoes […]
>
> – James Joyce, *Finnegan's Wake*

Formal languages are more **strict**—you have to follow rules *exactly*.

- For example, here's an invalid mathematical expression:
  - $3/+6\$$
- What makes it invalid?

Programming languages—like all formal languages—have specific:

- Tokens (use the right kinds of characters)
  - The $\$$ token doesn't have meaning in math.
- Structure (use tokens in the right order)
  - $3/+6$ has valid tokens but they're not in the right order.

Together, meaningful tokens + correct structure = ***syntax***

In order to write JavaScript programs that run successfully, we need to learn the rules for creating valid JavaScript syntax!

JavaScript Syntax

A preview of what's to come

```
const letters = 'abcde';
console.log(letters.length);

let i = 0;
while (i < 5) {
  console.log('Current letter:',
letters[i]);
  i = i + 1;
}

const isRaining = true;
```

- Each statement on one line, by itself, ending with a semicolon (`;`)
- Capitalization matters
- Curly brackets (`{}`) group code together
- Quotes around strings like `'All digits:'`—no quotes around numbers like `0`
- Special words like `while`, `let`

```
if (isRaining) {
  console.log("Don't forget your
umbrella");
}

console.log('Done!');
```

- …and more!

NOTE

Semicolons

Semicolons aren't strictly necessary for JavaScript code to work. If you don't add semicolons, JavaScript will attempt to guess where semicolons should go. This is called ***automatic semicolon insertion***. You won't break your code by forgetting a semicolon, but it's good practice to put a semicolon at the end of each line, especially since it'll help you write other C-like languages where semicolons are required (like C and Java).

A note on syntax errors

When you're learning a new language (even languages that aren't programming languages) it's normal to get it wrong all the time.

Programming languages like JavaScript are very strict and exact with syntax.

JavaScript tries to help by outputting a `SyntaxError` when it can't understand your code.

Here's a sentence that JavaScript won't understand (even though *we* can understand it just fine):

```
if $2.00 is pretty close to $2.50, then say "Close enough!"
```

If we try to run this in JavaScript…

```
if $2.00 is pretty close to $2.50, then say "Close enough!"
   ^^

Uncaught SyntaxError: Unexpected identifier '$2'
```

JavaScript will throw a ***SyntaxError*** if it can't understand your code

- It'll even try to guess the part of your code that created the error!
- Expect this to happen quite often—it's completely normal to have a lot of errors while learning
- **Tip:** read JavaScript's error messages and check for typos

The traditional first program

```
console.log('Hello, world!');
// JavaScript is super cool!
```

- `console.log` is a ***function*** used to display ***values*** placed inside the parentheses (`()`)
- `'Hello, world!'` is a value that's a ***string*** (as in, "a *string* of text")
- `//` marks a line as a ***comment***—JavaScript will ignore comments

`console.log`

The `console.log` function is a web developer's best friend!

```
console.log('Hi');
console.log();
console.log('How are you?');
```

Notice that `console.log()` shows up as an empty line.

Later, we'll use `console.log` to inspect variables and see how their values change.

Comments

Comments are great for annotating and adding notes to your source code.

```
console.log('Hi');
// TODO: eventually I should remove the line below...
// It's fine for now though.
console.log();
console.log('How are you?');
```

You can have multiple comments—as many as you want!

Values and Variables

Values and data types

A ***value*** is a piece of data.

Values belong to categories called ***data types***.

Different data types are treated differently by JavaScript.

Data types in JavaScript

Name	Example
String	`'Hello, world!'`, `'!'`
Number	`1`, `3.14`, `-20.34666`
Boolean	`true`, `false`
Null	`null`
Undefined	`undefined`

Variables

Variables are used to store values so you can use them later.

```
const greeting = 'Hello, world!';
console.log(greeting);
```

```
Hello, world!
```

Steps to create a variable:

1. Think of a good name for the variable (ex.: `favFruits`, `greeting`)
   - Valid characters are letters, numbers, and underscores
   - As long as you don't start with a number
2. Start with a ***declaration*** (`const`), then your variable name
3. Add the ***assignment operator*** (`=`) after the name
4. Add the value you want to store in the variable after `=`
   - Ex.: `const greeting = 'Hello, world!';`

You'll see lots of examples of creating and using variables throughout this lecture!

Variable reassignment

Sometimes, you'll want to *update* the value stored in a variable.

```
const score = 0;
```

```
// This won't work!
score = score + 10;
```

```
Uncaught TypeError: Assignment to constant variable.
```

To create variables that are re-assignable, use the `let` declaration:

```
let score = 0;

// Success!
score = score + 10;

console.log(score); // 10
```

NOTE

While it's possible to reassign a variable to a value of a different data type, it's not a good practice to do so.

`const` vs. `let`

```
// Must be initialized to a value
const name = 'Will';
console.log(name); // Will

// Doesn't work with reassignment
operator
name = name + 'iam'; // TypeError

// Can store any data type
const pi = 3.14;
```

```
// Can be initialized without a value
let currentID;
console.log(currentID); // undefined

// Works with reassignment operator
currentID = 105;

// Can store any data type
let shouldContinue = true;
```

Rule of thumb: prefer using `const` over `let` until you need a re-assignable variable.

NOTE

`var`

Earlier versions of JavaScript did not have `let` and `const`. Instead, variables were declared with the `var` keyword.

```
var myScore = 10;
```

`var` is currently outdated syntax (although it still works for backwards compatibility reasons). The way `var` works leads to unexpected behavior, which can cause bugs that are difficult to fix.

> All modern browsers now support `let` and `const` so you should never use `var`.

Basic Data Types

Overview of basic data types

In this section, we'll show examples of working with various data types:

- Strings
- Numbers
- Booleans
- `undefined`
- `null`

Strings

String basics

AKA a *string* of text.

A string that's an entire sentence

```
'Hello, world!'
```

A string that's just one character

```
'a'
```

A string with symbols and numbers

```
'! 3&*# 29'
```

Strings are surrounded by quotation marks.

You can use single quotes (`'`) or double quotes (`"`)

```
"Hello, world!"
```

Take care not to mismatch them!

```
// DON'T do this
"Hello, world!'
```

String concatenation

You can use the + (plus) operator to ***concatenate*** strings together.

```
let favWord = 'JavaScript';
favWord = favWord + '!!!';
console.log(favWord);

// You can concatenate multiple strings at once
console.log(favWord + ' ' + 'Yum.');
```

```
JavaScript!!!
JavaScript!!! Yum.
```

NOTE

Concatenate

Concatenate is a fancy word for "smush values together to form a new one". The act of adding one string to another with + creates a *brand new string* and can be referred to as ***string concatenation***.

Template strings

Another way to create a string is with a ***template string*** or ***template literal***.

Template strings are surrounded by backticks (`)

```
const phrase = `Gotta catch 'em all!`;
```

Template strings allow you to create multi-line strings.

```
const poem = `Roses are red
Sugar is sweet
His boots are too big
For his goshdarn feet`;
console.log(poem);
```

```
Roses are red
Sugar is sweet
His boots are too big
For his goshdarn feet
```

NOTE

Creating line breaks with \n

You can create strings that have line breaks using normal string syntax, too. Instead of using `Enter` to create a new line, you add the newline (`\n`) character:

```
const poem = 'Roses are red\nSugar is sweet\nHis boots are too big\nFor his goshdarn
feet';
```

They're also used to *template-in* values of JavaScript expressions.

```
const adjective = 'gigantic';
const color = 'brown';
const animal = 'fox';

const madlib = `The ${adjective} ${color} space hamster jumped over the lazy
${animal}.`
console.log(madlib);
```

```
The gigantic brown space hamster jumped over the lazy fox.
```

Other things you can do with strings

```
const word = 'coffee';

// Get length
word.length; // 6

// Get an individual character
word[0]; // 'c'
word[1]; // 'o'
word[2]; // 'f'

// Return a new string with uppercased letters
word.toUpperCase(); // 'COFFEE'

// Check if string has 'a' in it
```

```
word.includes('a'); // false

// Check if string starts with 'cof'
word.startsWith('cof'); // true
```

For a complete list, see MDN’s JavaScript Reference doc on strings.

Numbers

Number basics

Numbers can be positive or negative.

Positive, unsigned numbers

```
100
29.99
```

Negative, signed numbers

```
-150078
-0.5
```

Arithmetic operators

You can perform mathematical operations on numbers.

Name	**Operator**	**Example**
Add	`+`	`5 + 3`
Subtract	`-`	`5 - 100`
Multiply	`*`	`2 * 3`
Divide	`/`	`5 / 2`
Exponents	`**`	`5 ** 2`

For example:

Adding numbers together

```
const total = 100 + 0.5;
console.log(total);
```

```
100.5
```

Dividing numbers

```
const numPrizes = 20;
const totalAttendees = 5;

// Show prizes per attendee
console.log(numPrizes / totalAttendees);
```

```
4
```

Updating numbers

Common operation: reassign to update a number value.

```
let sum = 0;
console.log(sum);

sum = sum + 2;
console.log(sum);

sum = sum + 2;
console.log(sum);
```

```
0
2
4
```

Shortcut `+=`/`-=`

```
sum += 2; // sum = sum + 2

sum -= 1; // sum = sum - 1
```

Booleans

About boolean values

There are just two possible boolean values: `true` and `false`.

They're used to execute code based on ***conditional logic*** (more on this later)

Example of conditional logic

```
const sleepy = false;
if (sleepy) {
  console.log('Yawn...');
} else {
  console.log("I'm awake!");
}
```

```
I’m awake!
```

Syntax matters

- JavaScript is case-sensitive: `true` and `false` are *not* the same as `True` and `False`
  - What would happen if you tried to run `console.log(True)`?

    ```
    Uncaught ReferenceError: True is not defined
    ```

- `true` and false are also not equivalent to `'true'` and `'false'`
  - Which values are booleans? Which ones are strings? How can you tell?

Comparison expressions

Comparison expressions are used to compare one value with another and return a boolean value.

Is `0` greater than `200`?

```
console.log(0 > 200);
```

```
  false
```

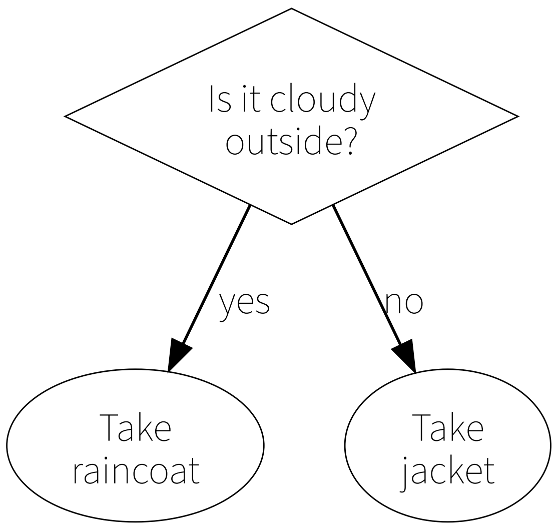
Here are the comparison operators in JavaScript.

Name	Operator	Example
Equals	`===`	`100 === 100`
Not equals	`!==`	`100 !== 100`
Less than	`<`	`100 < 100`
Greater than	`>`	`100 > 100`

Name	Operator	Example
Less than or equal to	`<=`	`100 <= 100`
Greater than or equal to	`>=`	`100 >= 100`

Conditional statements

Conditional logic (or branching logic) is when a program executes different procedures depending on a specified condition.



Booleans are used to implement conditional logic with the `if...else` statement.

Check if names are the same and ouput a message

```
const myName = 'Muir';
const yourName = 'Sam';

if (myName === yourName) {
  console.log('We have the same name!');
} else {
  console.log("That's a nice name.");
}
```

```
That’s a nice name.
```

Output an error message if password is too short

```
const password = 'hunter2';
if (password.length < 8) {
  console.log('Password is too short!');
}
```

```
Password is too short!
```

Compare number of pets

```
const myNumPets = 2;
const yourNumPets = 4;

if (myNumPets > yourNumPets) {
  console.log('I have more pets than you!');
} else if (myNumPets < yourNumPets) {
  console.log('You have more pets than me!');
} else {
  console.log('We have the same number of pets!');
}
```

```
You have more pets than me!
```

NOTE

Block statements

Block statements group multiple statements together inside curly brackets (`{}`).

`undefined` and `null`

- When you declare a variable but don't give it an initial value, its value will be set to `undefined`.
- Another value used to represent nothingness is `null`.
  - `null` is the *intentional* absence of value, whereas `undefined` means no value has been assigned yet.

Functions

About functions

You're already familiar with the `console.log()` function:

```
console.log('Hello, world!');
```

Functions are named groups of code that can be reused.

> **NOTE**
> **Methods**
>
> In JavaScript, a ***method*** is a type of function. You'll learn more about methods during the lesson on objects.

Calling functions

You can ***call*** a function by adding parentheses (`()`) after its name.

This will execute the block of code inside the function.

The parentheses are important! Without them, JavaScript won't call the function.

So, to call `console.log()`…

1. Type the function's name, `console.log`

   ```
   console.log
   ```

2. Follow it with a pair of parentheses (`()`)

   ```
   console.log()
   ```

3. List inputs (if any) *inside* the parentheses

   ```
   console.log('Hello, world!');
   ```

Looking Ahead

Coming up

- More about conditional logic
- and loops!