



CTD Intro Week 1

Programming Fundamentals:
JavaScript Basics

Programming Fundamentals

- Write programs with future maintainers in mind
 - Clarity, simplicity, comments
 - Most programming involves updating/fixing code someone else wrote
- Understand the problem
 - Inputs, outputs, user interface, steps from inputs to outputs
- High-level steps
 - Pseudocode
- Divide and conquer
 - Code and test simpler components which taken together solve the problem
- DRY – Don't Repeat Yourself
 - Capture reused components in one place
 - usually a function or method
 - Don't copy and paste



JavaScript

- Javascript is not java!
- Invented in 1995 at Netscape
 - Called javascript because java was new, popular and exciting
 - But it is unrelated to java
- Standardized as ECMAScript (ECMA-262) in 1997
 - European Computer Manufacturers Association (ECMA)
- Important revisions
 - ES5 (2009)
 - Var for non-global scope, function scope only
 - ES6 (2015)
 - Lexical (block) scope with let and const
 - class, module
 - Anonymous function shorthand, arrow notation (a, b) => { }
- Most popular computer language
- Lots of built-in capabilities
- Rich set of packages available
- Highly optimized, good performance
- In all browsers
- Node.js for servers and command line apps (Google javascript engine)





Some JavaScript facts

- First class functions
 - Functions can be assigned to variable (not true in every language!)
 - `myObject.func` returns the function
 - `myObject.func()` calls the function
- Very permissive
 - Doesn't, by default report errors on many things which are probably wrong
 - Automatic conversions between types
 - Doesn't check type or number of function arguments
 - Fills in with undefined values if necessary
 - Typescript was invented to fix this
- Convenient object model
 - Any mix of indexed arrays [...] and associative arrays {...}
 - Mixed datatypes
 - So simple and useful it became a data exchange standard
 - JavaScript Object Notation (JSON)

The 8 Data Types in JavaScript

- Number
 - Double precision floating point, also used to represent integers
- String
 - Characters (and anything representable by Unicode)
- Boolean
 - True/false
- Undefined
 - Its type is 'undefined', Lack of a value, never assigned, tests as false
- Null
 - Its type is 'object', absence of an object, tests as false
- Object
 - Combinations of indexed and associative arrays
- Symbol
 - Unique, immutable value to use as a key for objects
- BigInt
 - Integers with unlimited precision (subject to resource limitations)
- Use `typeof(<name>)` to find out what a variable's datatype is
 - Where `<name>` is a placeholder for any variable name or literal

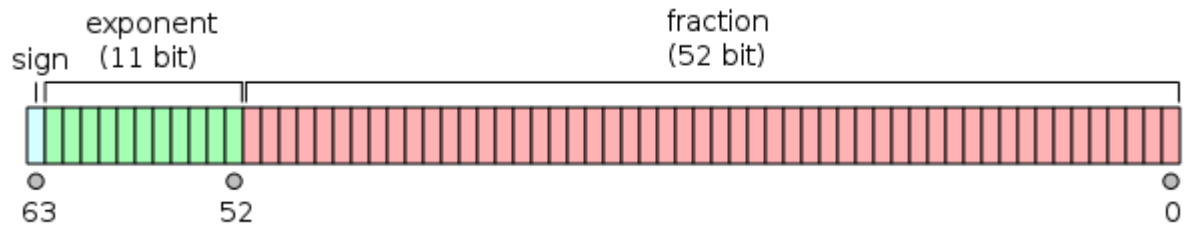


Behind the Scenes

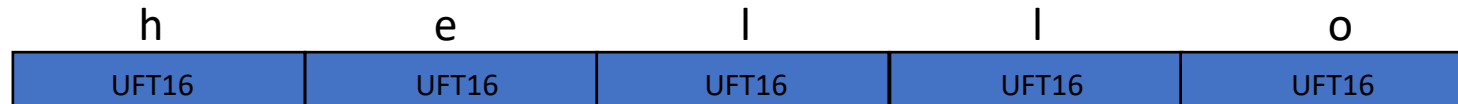
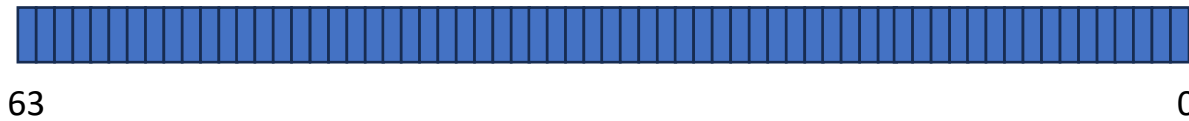
- Don't need to know details of compilation and data types for the intro class!
 - Context and background
- Different datatypes have varying internal representations
- Use explicit conversions between datatypes



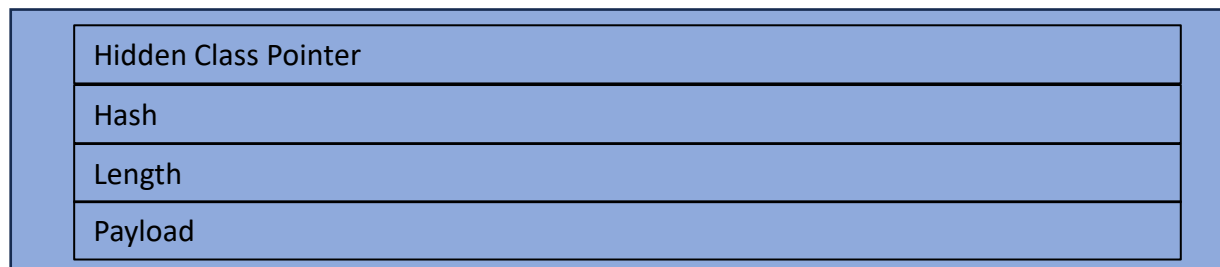
Data Types in JavaScript



[This Photo](#) by Unknown Author is licensed under [CC BY-SA](#)



Can also be UTF8, UTF32...



Object Header

Floating Point – Number in JavaScript

Integer – Not used for JavaScript Numbers

String – Characters in memory

Objects – combinations of indexed and associative arrays

Explicit Conversions

- `Number()`
- `String()`
- `parseInt()`
- `parseFloat()`
- `Boolean()`
- `<num>.toFixed(<decimalPlaces>)`
 - String with fixed formatting
- `Math.floor()`
- `Math.round()`



JavaScript Syntax

- [airbnb/javascript: JavaScript Style Guide \(github.com\)](https://airbnb/javascript/)
 - Common set of conventions for code
- Use lesson content for a comprehensive syntax guide
 - Just highlights in the presentation
- Variable naming
 - Use camelCase, longNameCamelCase
 - Use UPPERCASE constants to remember values (sometimes globally)
 - `Const FINESTRUCTURECONSTANT = 1.0/137.0;`
 - Case sensitive: `thisVar` is not the same as `thisvar`
 - Reserved words, can't use parts of the language as variable names
 - `let let = 5; //` won't work

JavaScript Syntax – Strings

- Literals
 - 'a string' and "a string" the same except
 - Need to escape quotes inside strings, so
 - "isn't" works without escaping
 - Vs 'isn\'t'
 - JSON only accepts ""
 - ``` allows interpolation of `${anyVar}` // string representation is inserted
 - Can also interpolate expressions
- Concatenate using `+`
 - `FooBar = 'foo' + bar;`
- Accessing individual characters
 - `charAt()` method or `[]`
- Length attribute (works for string and arrays)
 - `"abcde".length` will return 5

More Syntax

- Operator precedence
 - When in doubt, use () to group
- ';' at the end of every statement line
 - Not worth worrying about where they can be left out
- Functions and methods
 - Variables can contain functions (first class functions)
 - Call by adding (), can contain an argument list (arg1, arg2...)
 - aFunction is a variable, running it returns the function
 - Running aFunction() calls the function
 - Methods are functions which are part of an object and which act on it
 - Let num = 63;
 - Num.toFixed()

Variables and Scoping

- Variables give a name to a value
 - The value will have a data type
- Variable scoping
 - Defines the places a variable name is recognized
- Global scope
 - Dangerous and not advised
 - Valid everywhere, changing a global might impact code anywhere which references it
- Lexical scope
 - Value is local to a block { ... }
 - Preferred
 - Defined using let or const
- Function scope
 - Defined using var
 - Value is defined anywhere in the enclosing function
 - Obsolete in most cases. Use let and const.



Let and Const

- If you don't expect the value change, use const
 - For single values it can't be changed
 - For objects, it is a constant reference, which means it's the same data structure, but the content can be changed.
- Use let for everything else
 - Lexically (block) scoped
 - Scope the variable as narrowly as possible, but not too narrowly

```
let stableAcrossLoop = 1;
while(someTest()) {
  let newEachLoop = 0;
  stableAcrossLoop = someFunction(stableAcrossLoop, newEachLoop);
}
// newEachLoop is undefined here
// stableAcrossLoop contains the value returned by someFunction during the last loop
```

Truth and Equality

- Javascript does lots of automatic conversions
 - `==` can produce unexpected results
 - `0 == ''`
 - `null == undefined`
 - `[] == ![]`
 - `[2] == 2`
 - `\n == 0`
 - Almost always, use `===`, checks for same type and value
- What is truth?
 - true (false) – explicit
 - true: non-zero Number, non-empty string, any (even empty) object
- Logical operators `&&`, `||`, `!` To create logical expressions
 - Short circuit – only executes what is needed to determine the final value
 - `false && 'this never runs'`
 - `false || 'this does run'`
- Truth values are used in if/else statements and loops

```
if ([pair1[0] === pair2[0] && pair1[1] === pair2[1]]) {  
  return true;  
}  
else {  
  return false;  
}
```



```
// Anatomy of a function declaration
function aNewFunc(parm, anotherParm) { // parameter definitions
    // This is the function body, it has its own scope inside { }.
    // Computations are performed here in the body and the result is returned
    // A return statement can occur anywhere and multiple times.
    // Return exits the function and provides a value.
    // Return is not required, but if not specified, 'undefined' is returned.
    return parm + anotherParm;
}
```

Functions

- DRY Principal – Don't Repeat Yourself
- A way to encapsulate and reuse a piece of code
- Parameterized
 - Parameters define values which can be passed to the function
 - Function myFunc(parameter1, parameter2)
 - Parameters 1 and 2 can be referenced inside the function body {
 }
 - Arguments are the actual values used when a function is called
 - let result = myFunc(53, "myString");
- Returns a value using a 'return' statement

Defining Functions

- Named function declarations
- Anonymous function declarations
- Shorthand function declarations (arrow notation)
 - `function(a, b) {a + b}`
 - `(a, b) => {a + b}`
- Hoisting
 - Definitions are hoisted to the beginning of the scope
- Missing or extra parameters
 - Filled in with undefined unless defaulted
- Default values
 - `function(a = true, b = 63) { ... }`
- Can use `typeof()` to check arguments
 - `If (typeof(param1) !== 'string') {throw...`
- Throwing an error:
 - `throw new Error("an error message");`

```
// anonymous function assigned to a lexically scoped variable
let funcInAVar = function(oneParam, twoParam) {
  return hoistedFunc(oneParam * twoParam);
}

// functions can be passed as parameters
// It is common in JavaScript
// Shorthand notation was invented to make it less verbose

// anonymous function calling in setTimeout
for (var i = 0; i < 10; i++) {
  setTimeout(function() {
    console.log(i);
  }, 1000 * i);
}

// shorter and simpler with => notation
for (var i = 0; i < 10; i++) {
  setTimeout(() => console.log(i) , 1000 * i);
}

// This named function can be called before it is defined.
// This is called hoisting.
function hoistedFunc(param) {
  return Math.random() * param;
}

// this function has defaulted parameters
function defParam(p1 = 53, p2 = "Tom") {
  return `${p2}'s favorite number is ${p1}`;
}
```

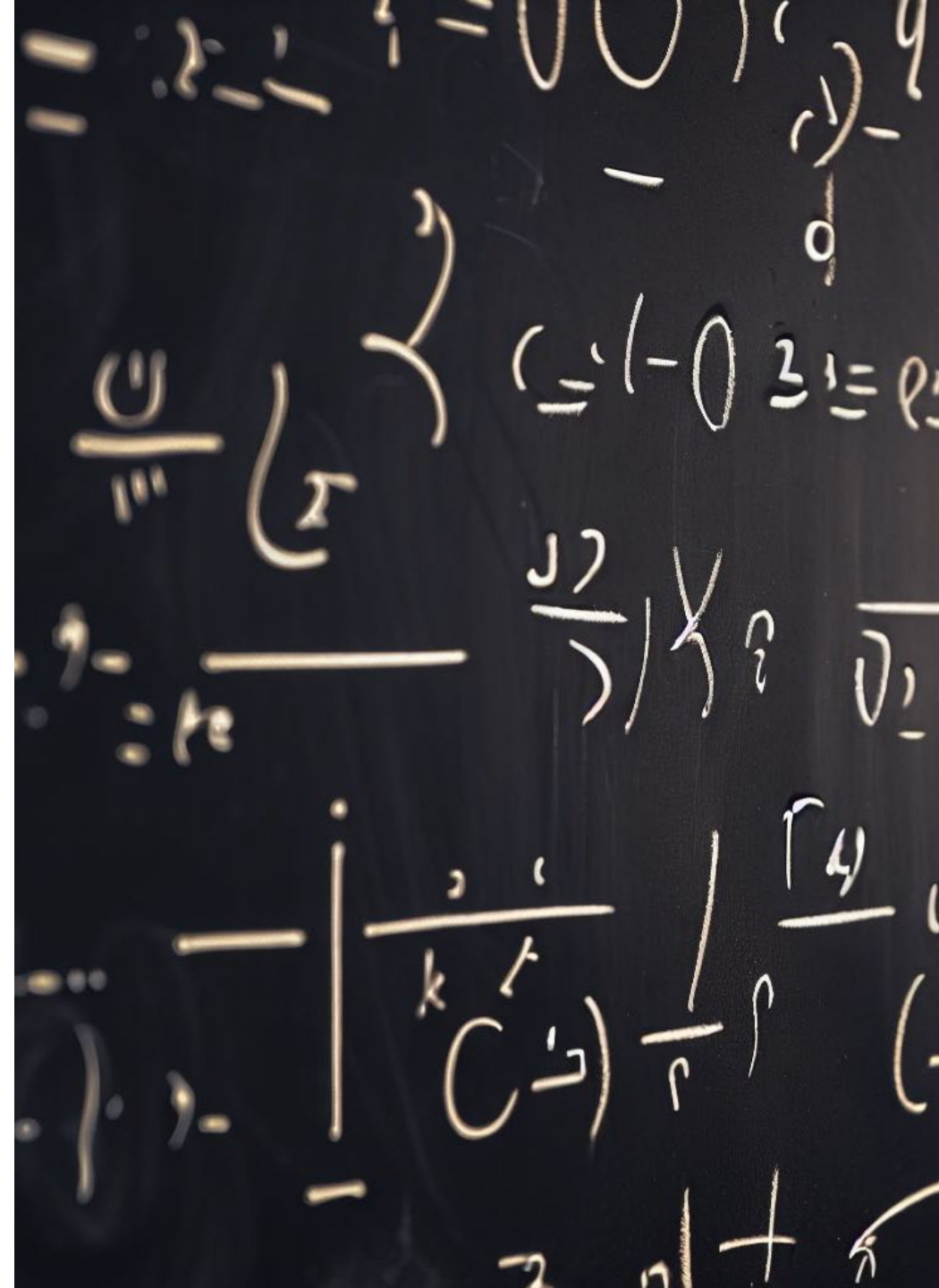
Dates and Times

- Unix time (epoch)
 - The number of seconds since 1/1/1970
 - `Date.now()` returns the number of milliseconds since 1/1/1970
- Date module
 - `let today = new Date()` creates a date object at the current date and time
 - `today.getTime()` unix epoch at the time 'today' was created
 - `getFullYear()`, `getMonth()`, `getDate()`, `getDay()`
 - `getHours()`, `getMinutes()`, `getSeconds()`, `getMilliseconds()`



The Math module

- `Math.random()`
 - Returns a floating point number from zero to 1
 - Zero is included, 1 is excluded (0.0 -> 0.9999999999)
 - Often scaled to create a random integer in a range
 - `Math.floor(scaleInt * Math.random()) + 1` // integer in the range 1 -> scaleInt
- `Math.floor(num)` // next lower integer, remove fractional part
- `Math.round(num)` // rounds up or down depending on ≥ 0.5
- `Math.abs(num)` // absolute value
- `Math.max(n1, n2, ...)` // maximum of the list of numbers



Coding Assignment

- The coding assignment is at the bottom of each lesson page

Click here to go to the codesandbox or github lesson

Assignments

Due Date: Feb 25, 2025

Coding Assignment

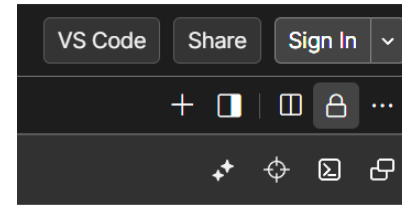
The coding assignment for this lesson can be found [here](#)

Submit Assignment

Click here for the assignment submission form (bottom right)

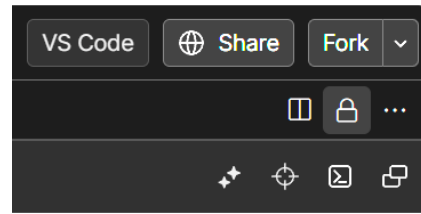
Using CodeSandBox

- Login using google or github:



- Fork the lesson

- Make your own copy

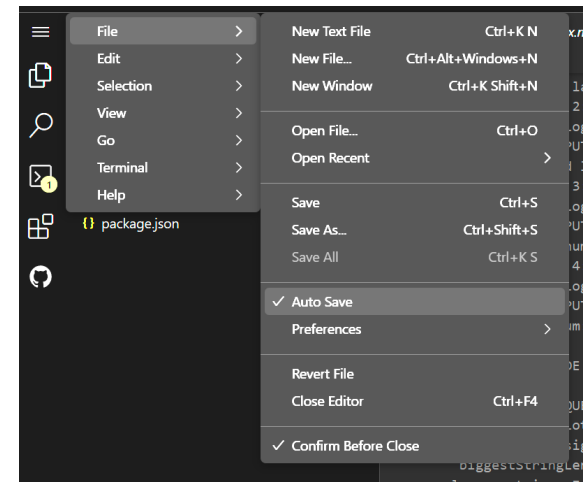
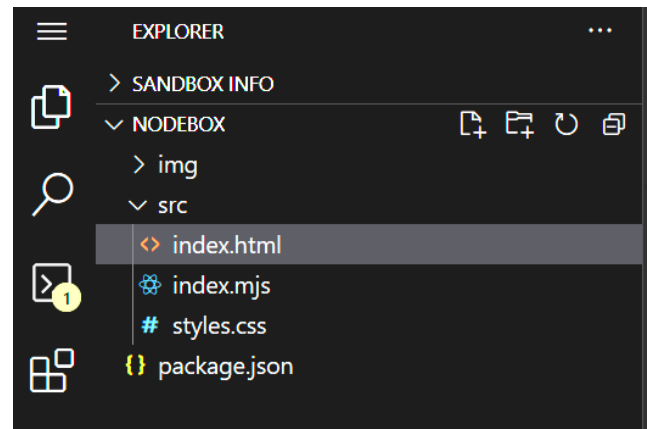


- Editing window

- Central window where you write your code
- Turn on word wrap (alt-z on Windows, cmd-z on Mac, or in the menu view->word wrap)
- Check/turn on autosave in the file menu
- Edit index.mjs by clicking on it

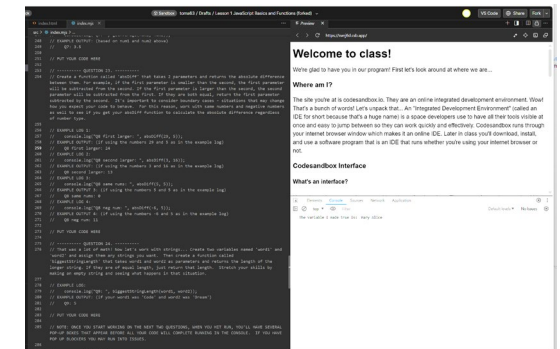


- File browser

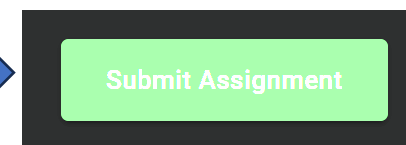
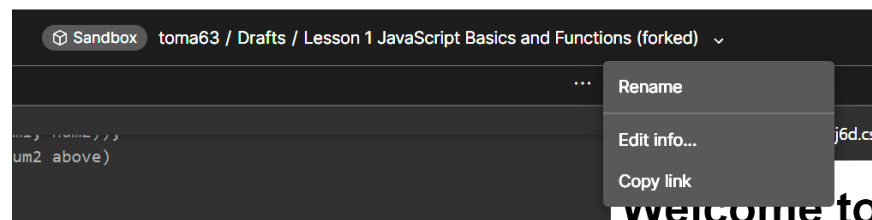


Using CodeSandBox

- Open devtools to see console logs
- The console appear in the lower pane on the right



- The console updates are live, so you may see errors while typing
- Once you are done with the lesson and all results have been logged to the console, paste your forked sandbox into the lesson submission form.



Link to your Coding Assignment / Pull Request / Final Project Repository *

For Intro students, weeks 1 - 6, please paste the URL for your repl.it.com work.

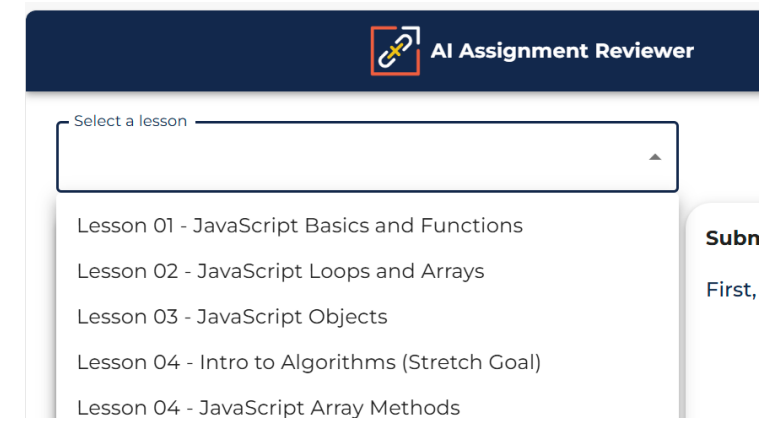
For Intro students, week 7 and later
AND
Rails, React, or Node students, please paste the URL for your github.com pull request.

CodeSandBox Video

- Here's a video explaining how to use CodeSandBox
- [CodeSandBox Video](#)

AI Reviewer

- Quick way to get detailed feedback
- Experimental
 - Check with Mentors or CILs if you think it's incorrect
 - Can file issues using the bug icon
- Your official code reviews are done by a human mentor
- Access it here: [AI Assignment Reviewer](#)
- Select your lesson in the dropdown
- Paste your code into the middle pane
- You can ask follow-up questions in the chat
 - Right side pane



Debugging

- Find out what the variables contain
- Test the functions and methods individually
- `Console.log()`
 - In codesandbox, goes to the console tab/window in the devtools window
 - In a browser, goes to developer's tools console
- Developer's tools
 - Debugger, console
 - Lots of other goodies
 - Coming in the debugging lesson
 - Not needed for codesandbox based lessons



Demo and Q&A

