The JavaScript Language (Part 1)

Robert M. Dondero, Ph.D.
Princeton University

Objectives

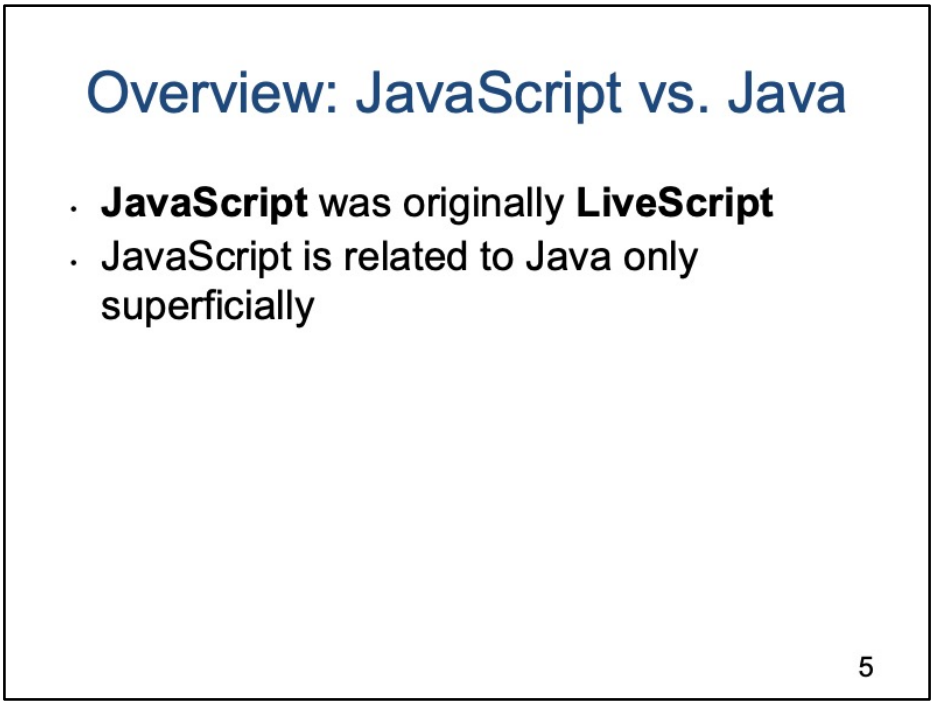
- We will cover:
  - A subset of JavaScript…
  - That is appropriate for COS 333…
  - Through example programs

Agenda

- **Overview**
- Setup
- Simple programs
- Functions
- Standard library
- Data types and operators
- Terminal I/O
- Exceptions
- Statements

Overview

- **Who**: Brendan Eich
- **Where**: Netscape
- **When**: 1995
- **Why**: Client-side scripting language for web pages



Overview: JavaScript vs. Java

JavaScript was originally LiveScript

Later changed to JavaScript to capitalize on popularity of Java

JavaScript is related to Java only superficially

Overview: Motivation

- **Question**:
  - Why study JavaScript?
- **Answer**: …

Overview: Motivation

Rank	Language	Ratings
1	C	16.5%
2	Java	15.1%
3	Python	9.0%
4	C++	6.2%
5	C#	5.3%
6	Visual Net Basic	5.2%
7	**JavaScript**	2.5%
8	R	2.4%
9	PHP	1.9%
10	Swift	1.4%
11	SQL	1.4%

https://www.tiobe.com/tiobe-index/ on 7/7/20

Overview: Motivation

Website	Front End	Back End
Google.com	**JavaScript**	C, C++, Go, Java, Python
Facebook.com	**JavaScript**	Hack, PHP, Python, C++, Java, Erlang, D, Xhp, Haskell
YouTube.com	**JavaScript**	C, C++, Java, Python, Go
Yahoo	**JavaScript**	PHP
Amazon.com	**JavaScript**	Java, C++, Perl
Wikipedia.org	**JavaScript**	PHP, Hack
Twitter.com	**JavaScript**	C++, Java, Scala, Ruby
Bing	**JavaScript**	ASP.net
eBay.com	**JavaScript**	Java, JavaScript, Scala
MSN.com	**JavaScript**	ASP.net

https://en.wikipedia.org/wiki/Programming_languages_used_in_most_popular_websites

Overview: ECMAScript

- JavaScript is an implementation of…
- ***ECMAScript***
  - By **E**uropean **C**omputer **M**anufacturer's **A**ssociation
  - Trademarked specification
  - Other implementations:
    - JScript (Microsoft)
    - ActionScript (Macromedia Inc)

Overview: ECMAScript

Year	Name	Description
1997	*ECMAScript1* *ES1*	First edition
1998	*ECMAScript2* *ES2*	Editorial changes only
1999	*ECMAScript3* *ES3*	Added regular expressions **Added try/catch**
	ECMAScript4 *ES4*	Never released

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

Overview: ECMAScript

Year	Name	Description
2009	*ECMAScript5* *ES5*	**Added "strict mode"** Added JSON support Added String.trim() Added Array.isArray() Added Array Iteration Methods
2011	*ECMAScript5.1* *ES5.1*	Editorial changes
2015	*ECMAScript6* *ECMAScript2015* *ES6*	**Added let and const** **Added default parameter values** Added Array.find() Added Array.findIndex() **Added classes** **Added promises**

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

Overview: ECMAScript

Year	Name	Description
2016	*ECMAScript7* *ECMAScript2016* *ES7*	Added exponential operator (**) Added Array.prototype.includes
2017	*ECMAScript8* *ECMAScript2017* *ES8*	Added string padding Added new Object properties **Added Async functions** Added Shared Memory
2018	*ECMAScript9* *ECMAScript2018* *ES9*	Added rest / spread properties Added Asynchronous iteration Added Promise.finally() Additions to RegExp

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

Overview: ECMAScript

Year	Name	Description
2019	*ECMAScript10* *ECMAScript 2019* *ES10*	Added Array.prototype.flat Added Array.prototype.flatMap Changed Array.sort and Object.fromEntries
2020	*ECMAScript11* *ECMAScript2020* *ES11*	Added BigInt Added null coalescing syntax

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

We'll study the most recent version,
but mostly ECMAScript6 / ECMAScript2015 / ES6

Overview: Transpiling

- **Problem**:
  - You create a script that uses some ES6 syntax
  - You want your script to be runnable on all browsers
  - Some old-but-still-widely-used browsers can interpret only ES5 syntax

Overview: Transpiling

- **Solution**: ***Transpile***
  - You transpile your script from ES6 to ES5
  - You provide your script to browsers as ES5
  - Example transpiler: ***Babel***
    - You even can transpile to ES3 if necessary
- JavaScript community uses transpilers heavily

Overview: Polyfilling

- **Problem**:
  - You create a script that uses ES5 syntax, but also uses some new modules – modules that are available only on new (post-ES5) browsers
  - You want your script to be runnable on ES5 browsers

Overview: Polyfilling

- **Solution**: ***Polyfill***
  - You polyfill those new modules
  - You compose your own versions of those new modules and include them in your code
  - Or better…
  - You include in your code an already vetted set of polyfills
- JavaScript community uses polyfills heavily

Overview: Learning JavaScript

- **Commentary**
  - All versions of ECMAScript are backward compatible
    - ES*n* interpreter can handle ES*n-1*, …, ES1 code
  - When pgmmers compose ES*n* code, they sometimes use syntax/features of ES*n-1*, ES*n-2*, …, ES1 that have been superseded by features of ES*n*

Overview: Learning JavaScript

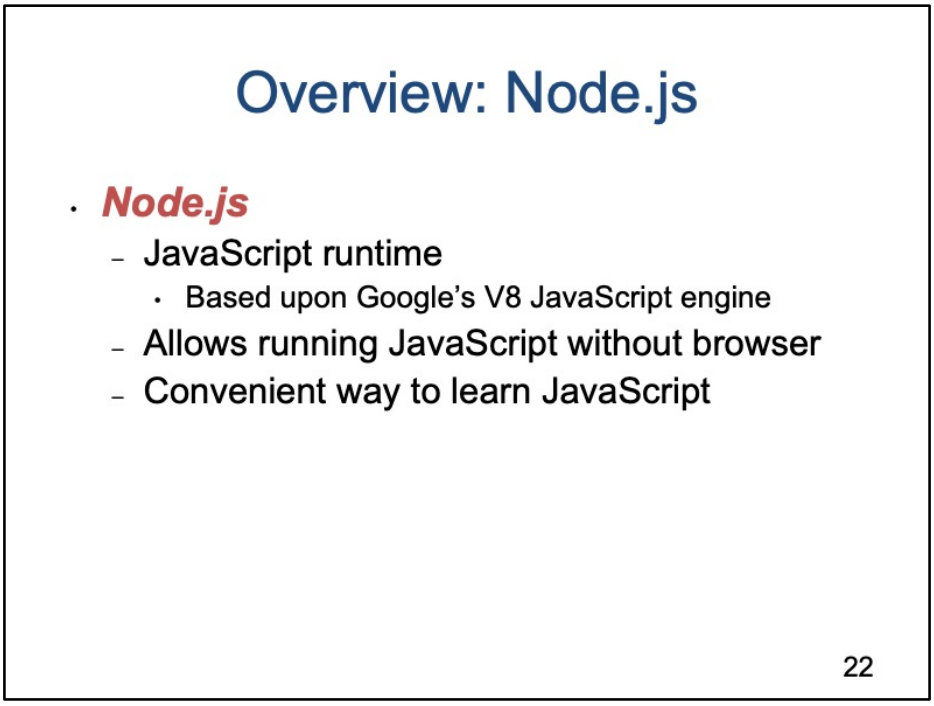
- **Commentary** (cont.)
  - To learn ES*n*, you must also learn ES*n-1*, ES*n-2*, …, ES1
  - JavaScript is a particularly difficult language to master

Overview: Running JavaScript

- Options for running JavaScript:
  - In browser (soon)
  - Via ***Node.js*** (now)

Overview: Node.js

- **Who**: Ryan Dahl
- **When**: 2009
- **Why**: Allow JavaScript to be used for **server-side** scripting



Overview: Node.js

Node.js
- JavaScript runtime
- Based upon Google's V8 JavaScript engine
- Allows running JavaScript without browser
  - Allows running JavaScript on server side of a web app
  - Allows "one language" apps
  - Has a HTTP server module
- Convenient way to learn JavaScript

Agenda

- Overview
- **Setup**
- Simple programs
- Functions
- Standard library
- Data types and operators
- Terminal I/O
- Exceptions
- Statements

Setup

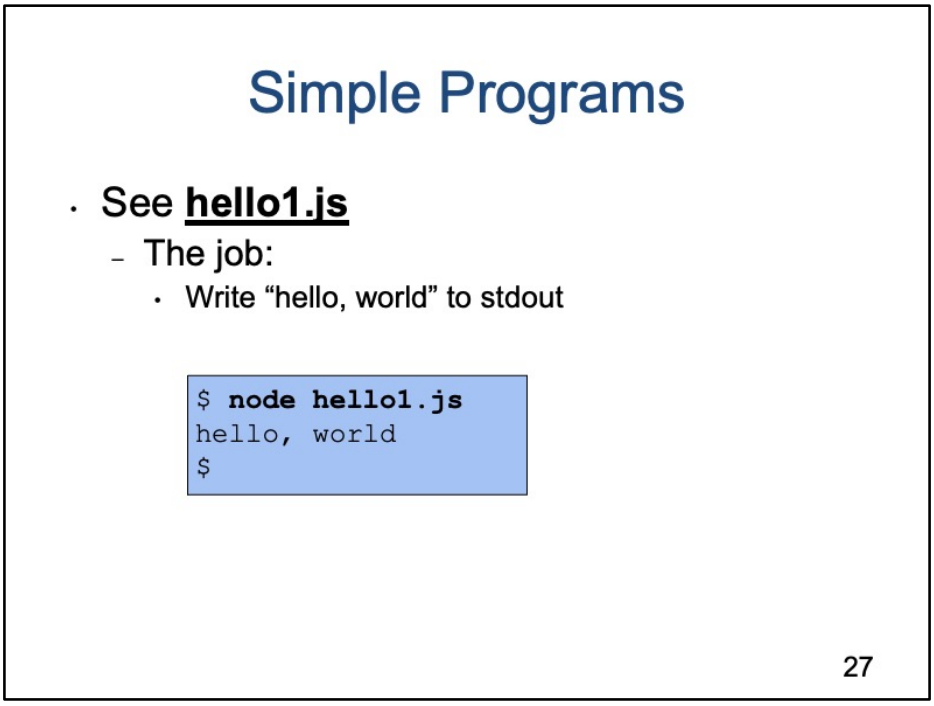
- **Step 1**: Install Node.js and npm
  - Linux
    - Use your package manager to install the node and npm packages
  - Mac
    - Use Homebrew to install Node.js by issuing this command:
      - `brew install node`
  - MS Windows
    - Browse to https://nodejs.org/en/download
    - Download an appropriate .msi file
    - In Windows Explorer, double click on the .msi file
    - Use the installation defaults

Setup

- **Step 2**: Install Node.js modules
  - At shell prompt
    - cd to the directory that contains JavaScript code
    - Install the `readline-sync` module
      - `npm install readline-sync`

Agenda

- Overview
- Setup
- **Simple programs**
- Functions
- Standard library
- Data types and operators
- Terminal I/O
- Exceptions
- Statements

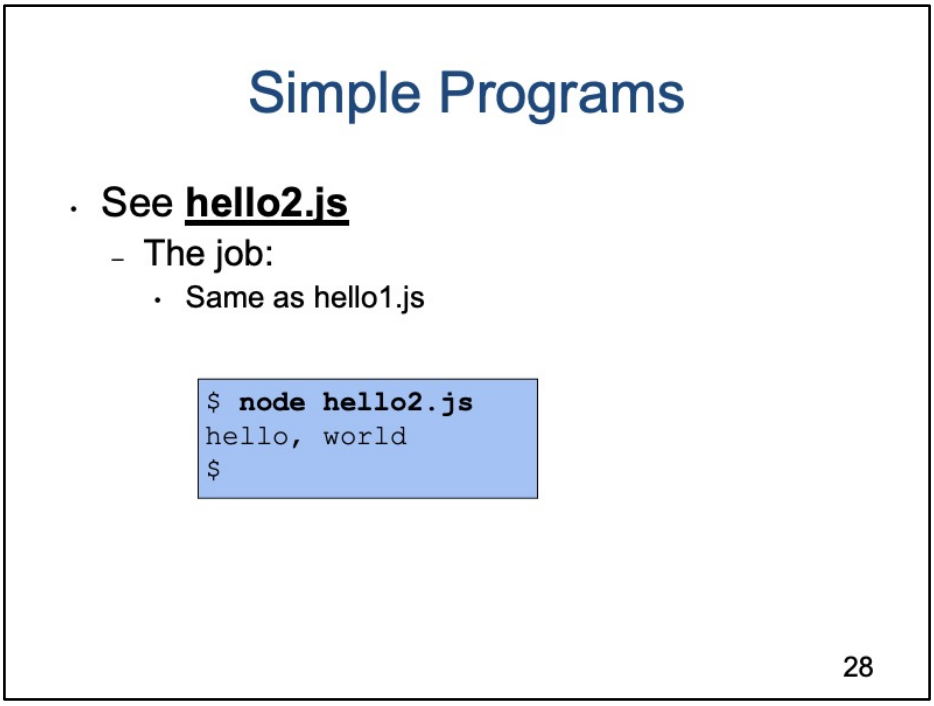


Simple Programs

[see slide]

Code notes

- The 'use strict' directive
  - Commands Node.js to use ***strict mode***
  - Requires variables to be declared (and more)
  - Without strict mode, undeclared variables are OK, and become global!!!
  - Automatic within class defs, modules
  - **Recommendation: always use strict mode**
- String literals
- Function calls
- The process.stdout.write() function
  - Alternative: the console.log() function
- Semicolons are optional
  - JavaScript does automatic semicolon insertion, following a set of ruies
  - I prefer not to know those rules!



Simple Programs

[see slide]

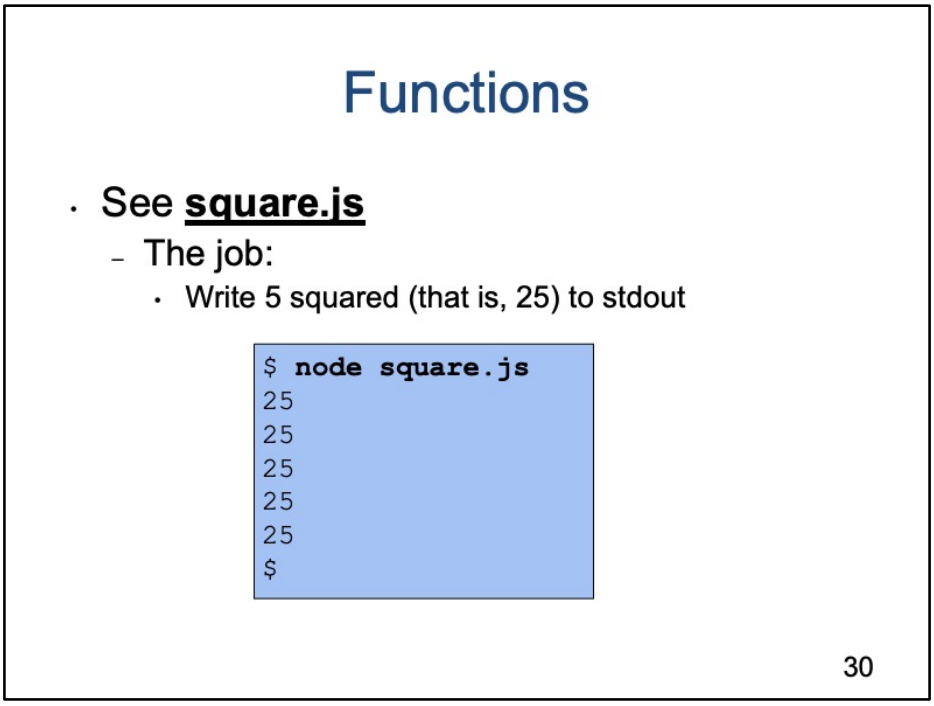
Code notes:

Avoiding global code

require.main === module is a feature of Node.js, not JavaScript

Agenda

- Overview
- Setup
- Simple programs
- **Functions**
- Standard library
- Data types and operators
- Terminal I/O
- Exceptions
- Statements

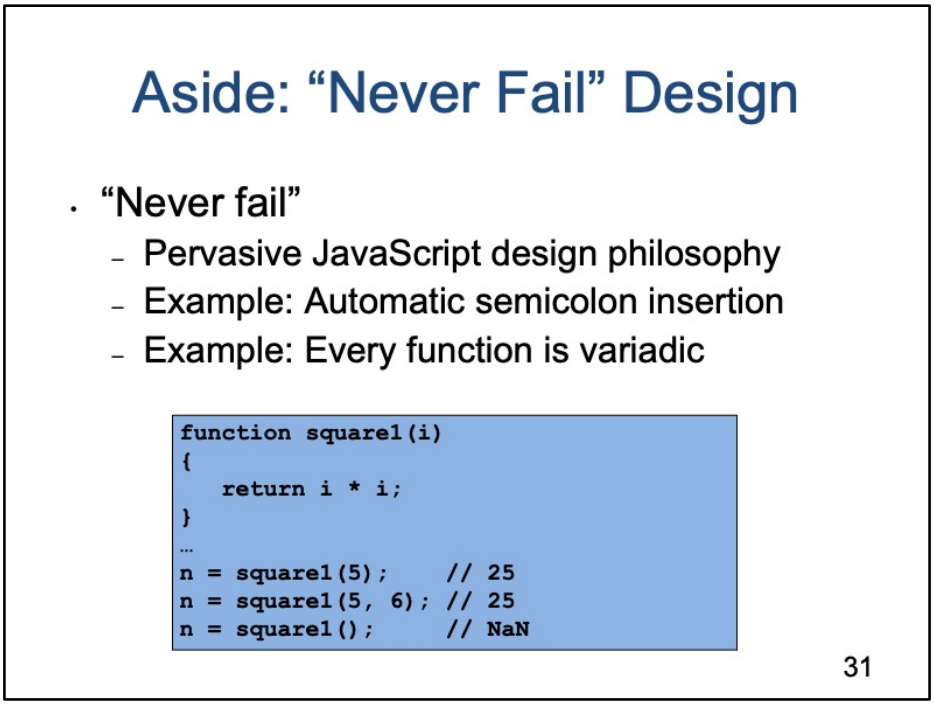


Functions

[see slide]

Code notes:

- let statement
  - Declares a variable
  - Optionally initializes a variable
- String() conversion function
- Function definition statement
- Function definition expression
- Arguments and parameters
  - Call is by value
- Arrow function definition expression
  - Described later



Aside: “Never Fail” Design

[see slide]

node.js doesn’t even generate a warning!!!

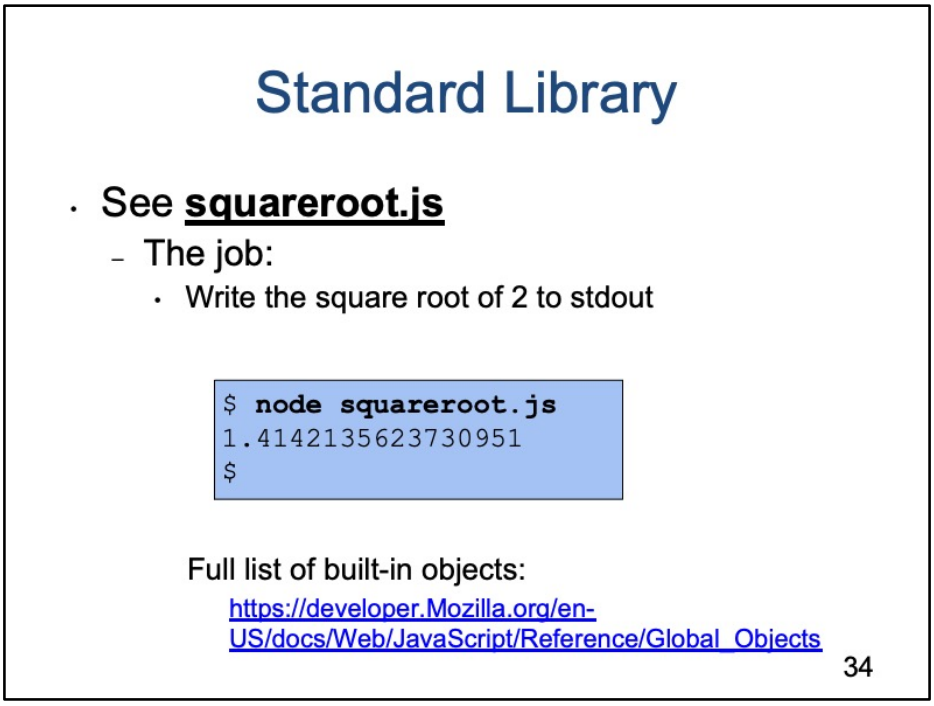
Aside: The Arguments Array

- The `arguments` array

```
function sumall()
{
   let sum = 0;
   for (let i = 0; i < arguments.length; i++)
      sum += arguments[i];
   return sum;
}
…
n = sumall();          // 0
n = sumall(5);         // 5
n = sumall(5, 6, 7);   // 18
```

Agenda

- Overview
- Setup
- Simple programs
- Functions
- **Standard library**
- Data types and operators
- Terminal I/O
- Exceptions
- Statements



Standard Library

[see slide]

Code notes:

Built-in Math object

Its sqrt() function

Built-in objects are available automatically

No import is necessary

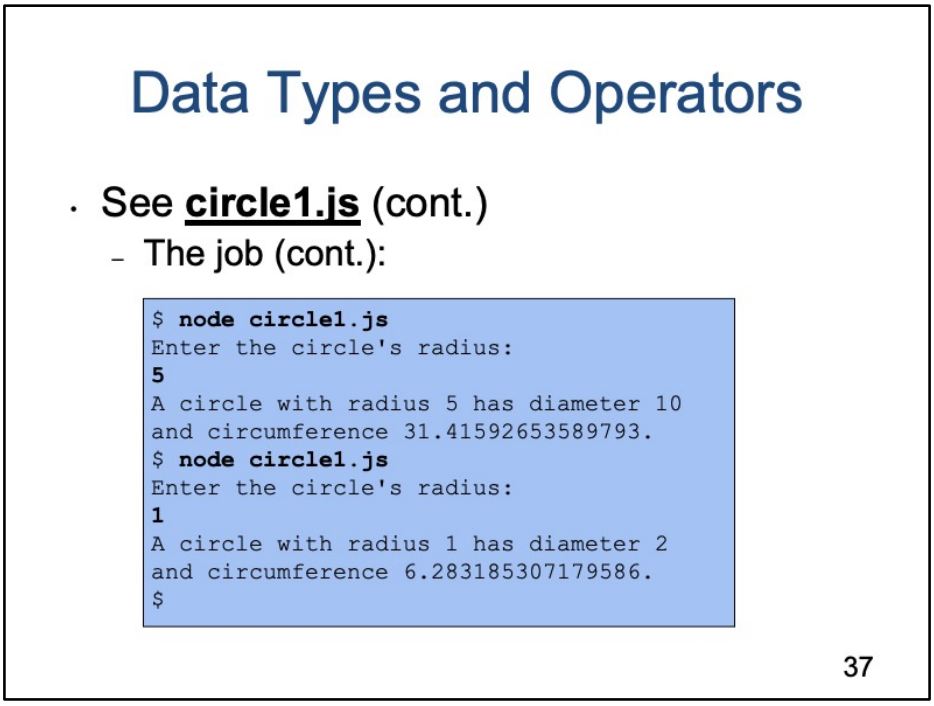
Full list of built-in objects:
https://developer.Mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects

Agenda

- Overview
- Setup
- Simple programs
- Functions
- Standard library
- **Data types and operators**
- **Terminal I/O**
- Exceptions
- Statements

Data Types and Operators

- See **circle1.js**
  - The job:
    - Read a circle's radius from stdin
    - Write the circle's diameter and circumference to stdout



Data Types and Operators

[see slide]

Code notes

- "Including" a Node.js module
- The readline module
- Terminal input via readline.question()
- Data types: Number, String
- Type conversions
  - Number() and String() functions
- Declaration (let) statements
- Local variables
- Operators: =, *, +

Generalizing…

Data Types and Operators

Data Type	Size	Example Literals
Boolean	1 byte	`false, true`
String	(varies)	`"hi", 'hi'`
Number	8 bytes	`123, -123, 0.0, 1.23, 1.23e4`

All numbers are stored as IEEE 754 floating point

38

Data Types and Operators

[see slide]

All Numbers are stored as floating point!!!
Specifically, in IEEE 754 format
Same representation as
- C double
- Java double
- Python float

Data Types and Operators

- Type conversion functions
  - `Number(x)`
  - `Boolean(x)`
  - `String(x)`
- Never fail! Examples:
  - `Number('') => 0`
  - `Number('xyz') => NaN`

Data Types and Operators

- Strong suggestion
  - Use type conversion functions to avoid mixed-type expressions!

If you need a laugh see:
https://www.destroyallsoftware.com/talks/wat

Data Types and Operators

Operators	(Priority) Meaning
`(…)`	(1) Grouping
`x.y`	**(2) Member access**
`x[y]`	**(2) Computed member access**
`new …`	**(2) New**
`f(…)`	(2) Function call
`x?.y`	(2) Optional chaining
`new`	(3) New without operator list
`x++`	(4) Postfix increment
`x--`	(4) Postfix decrement

Data Types and Operators

Operators	(Priority) Meaning
`!x`	(5) Logical NOT
`~x`	(5) Bitwise NOT
`+x`	(5) Unary plus
`-x`	(5) Unary negation
`++x`	(5) Prefix increment
`--x`	(5) Prefix decrement
`typeof x`	(5) typeof
`void x`	(5) void
`delete x`	(5) delete
`await x`	(5) await
`x**y`	(6) Exponentiation

Data Types and Operators

Operators	(Priority) Meaning
`x*y`	(7) Multiplication
`x/y`	(7) Division
`x%y`	(7) Remainder
`x+y`	(8) Addition
`x-y`	(8) Subtraction
`x<<y`	(9) Bitwise left shift
`x>>y`	(9) Bitwise right shift
`x>>>y`	(9) Bitwise unsigned right shift
`x<y`	(10) Less than
`x<=y`	(10) Less than or equal to
`x>y`	(10) Greater than
`x>=y`	(10) Greater than or equal to
`x in y`	(10) In
`x instanceof y`	(10) Instance of

Data Types and Operators

Operators	(Priority) Meaning
x==y	**(11) Equality**
x!=y	**(11) Inequality**
x===y	**(11) Strict equality**
x!==y	**(11) Strict inequality**
`x&y`	(12) Bitwise AND
`x^y`	(13) Bitwise exclusive OR
`x|y`	(14) Bitwise OR
`x&&y`	(15) Logical AND
`x||y`	(16) Logical OR
`x??y`	(17) Nullish coalescing operator
`x?y:z`	(18) Conditional

Data Types and Operators

Operators	(Priority) Meaning
`x=y, x+=y, x-=y, x**=y, x*=y, x/=y, x%=y, x<<=y, x>>=y, x>>>=y, x&=y, x^=y, x|=y, x&&=y, x||=y, x??=y`	(18) Assignment
`yield x`	(19) Yield
`yield* x`	(20) Yield
`x,y`	(21) Sequence

Data Types and Operators

Expression	Value
`123 == '123'`	`true`
`123 == '+123'`	`true`
`123 === '123'`	`false`
`123 === '+123'`	`false`

Equality operator (==) does type conversion

Strict equality operator (===) suppresses type conversion

Recommendation: Always use === instead of ==

Recommendation: Never mix types!!!

Terminal I/O

Reading from stdin:

```
const readlineSync = require('readline-sync');
…
str = readlineSync.question(str);
str = readlineSync.question();
```

Terminal I/O

Writing to stdout:

```
process.stdout.write(str);
console.log(expr);
```

Writing to stderr:

```
process.stderr.write(str);
console.error(expr);
```

Agenda

- Overview
- Setup
- Simple programs
- Functions
- Standard library
- Data types and operators
- Terminal I/O
- **Exceptions**
- Statements

Exceptions

- Recall circle1.js

```
$ node circle1.js
Enter the circle's radius:
5
A circle with radius 5 has diameter 10
and circumference 31.41592653589793.
$ node circle1.js
Enter the circle's radius:
xyz
A circle with radius NaN has diameter NaN
and circumference NaN.
$ node circle1.js
Enter the circle's radius:
A circle with radius 0 has diameter 0
and circumference 0.
$
```

Exceptions

- See **circle2.js**
  - The job:
    - Same as circle1.js, but also…
    - Handles bad data

Exceptions

- See **circle2.js** (cont.)
  - The job (cont.):

```
$ node circle2.js
Enter the circle's radius:
5
A circle with radius 5 has diameter 10
and circumference 31.41592653589793.
$ node circle2.js
Enter the circle's radius:
xyz
Error: Not an integer
$ node circle2.js
Enter the circle's radius:
Error: Missing integer
$
```

52

Exceptions

[see slide]

Code notes:

try… catch statement
"Never fail" philosophy
throw statement

Generalizing…

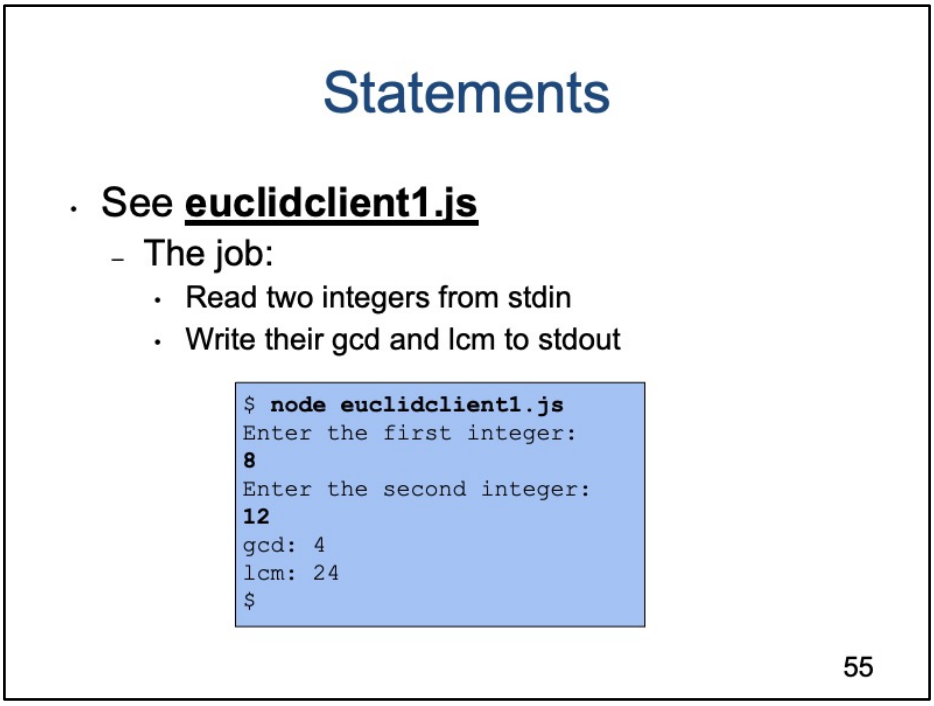
Exceptions

JavaScript standard exceptions

```
Error
     AssertionError
     RangeError
     ReferenceError
     SyntaxError
     TypeError
     SystemError
```

Agenda

- Overview
- Setup
- Simple programs
- Functions
- Standard library
- Data types and operators
- Terminal I/O
- Exceptions
- **Statements**



Statements

[see slide]

Code notes

Control statements: if, while, return, throw, try…catch

Generalizing…

Statements

Compound statement

```
{
    statement1;
    statement2;
    …
}
```

Statements

Variable definition statements

```
let name = expr;
const name = expr;

// Pre-ES6
var name = expr; // name is "hoisted"
```

Aside: Hoisting

```
function f()
{
   … // Not OK to use n here.
   let n = 5;
   … // OK to use n here.
}
```

```
function f()
{
   … // OK to use n here.
   … // But its value is undefined.
   var n = 5;
   … // OK to use n here.
}
```

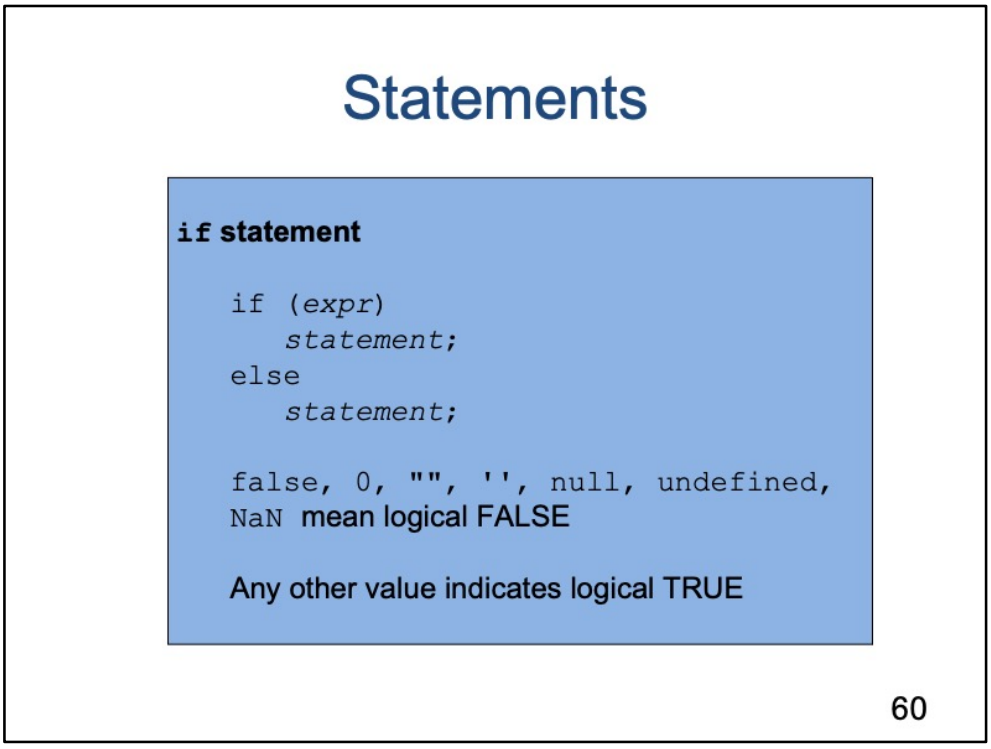
Statements

Function call statement

```
f(expr, expr, ...);
```

`return` statement

```
return;
return expr;
```



Statements

[see slide]

These are semantically identical:
if i == 0: …
if not i: …

These are semantically identical:
if len(somelist) == 0: …
if not somelist: …

Statements

`while` statement

```
while (expr)
   statement;
```

`false, 0, "", '', null, undefined, NaN` mean logical FALSE

Any other value indicates logical TRUE

Statements

do…while statement

```
do
   statement;
while (expr);
```

`false, 0, "", '', null, undefined, NaN` mean logical FALSE

Any other value indicates logical TRUE

Statements

`for` statements (by example)

```
for (let i = 0; i < 10; i++)
   statement;
for (let i = 0; i < someArray.length; i++)
   …someArray[i]…
for (let element of someArray)
   …element…
for (let property in someObject)
   …property…
```

Statements

`break` statement

```
break;
```

`continue` statement

```
continue;
```

Statements

try statement

```
try
   statement;
catch (exception)
   statement;
```

throw statement

```
throw object;
```

Agenda

- We have covered:
  - Overview
  - Setup
  - Simple programs
  - Functions
  - Standard library
  - Data types and operators
  - Terminal I/O
  - Exceptions
  - Statements