Front End
Software Development

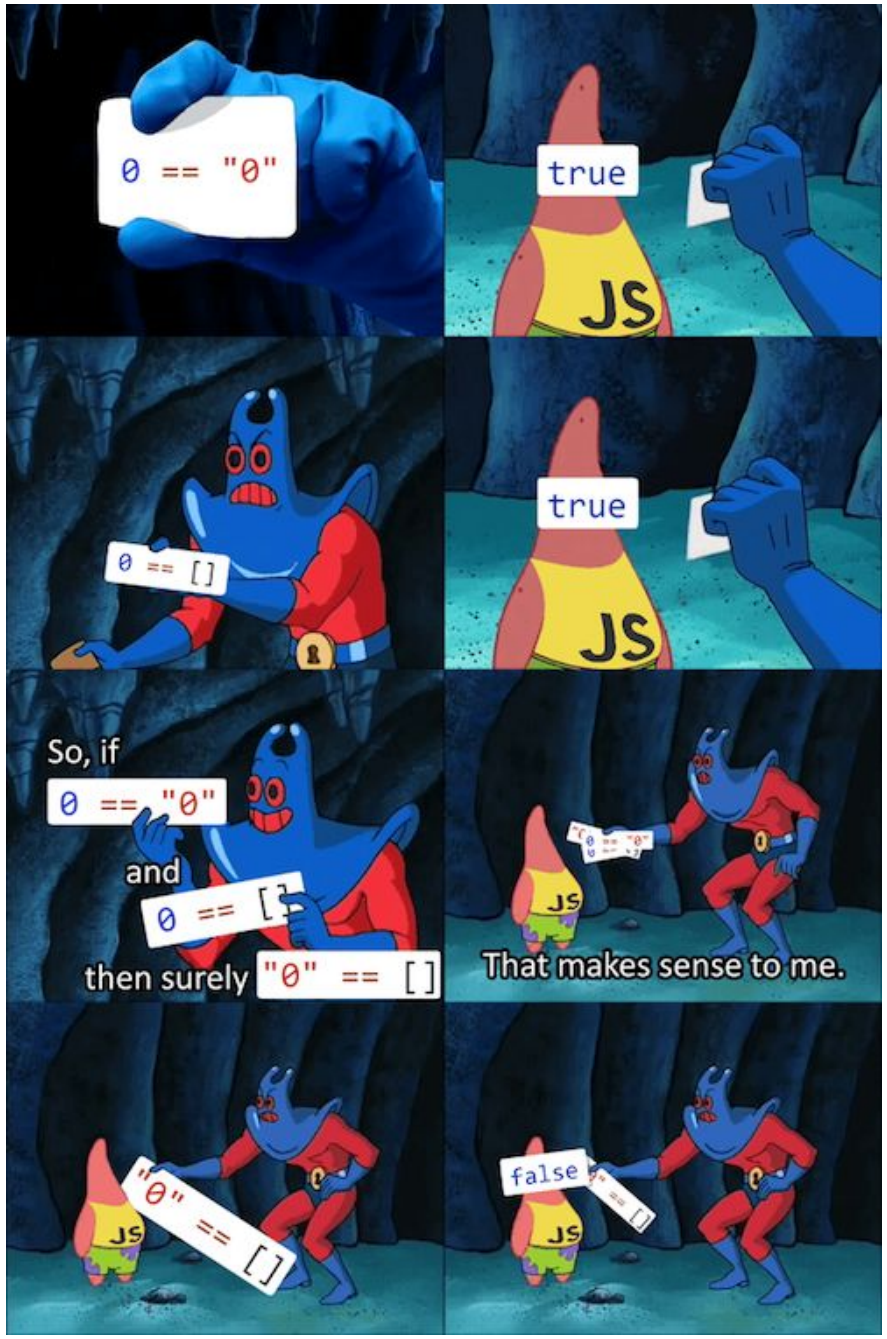
Introduction to JavaScript (weeks 1 - 6)

Week 04

Agenda

- Questions
- ECMAScript 6
  - `const` and `let`
  - Template Literals
  - Arrow Functions
- Callbacks
- Promises

Questions


0 == "0"
true
JS
0 == []
true
JS
So, if
0 == "0"
and
0 == []
then surely "0" == []
That makes sense to me.
"0" == []
false == []

const & let

(ECMAScript 6)

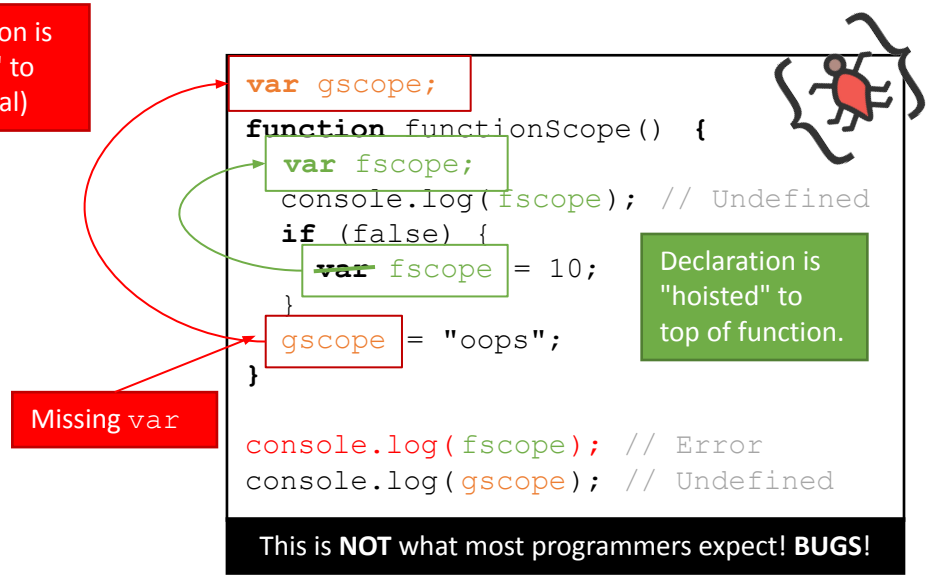
Syntactic Sugar (ES6 / ES2015)

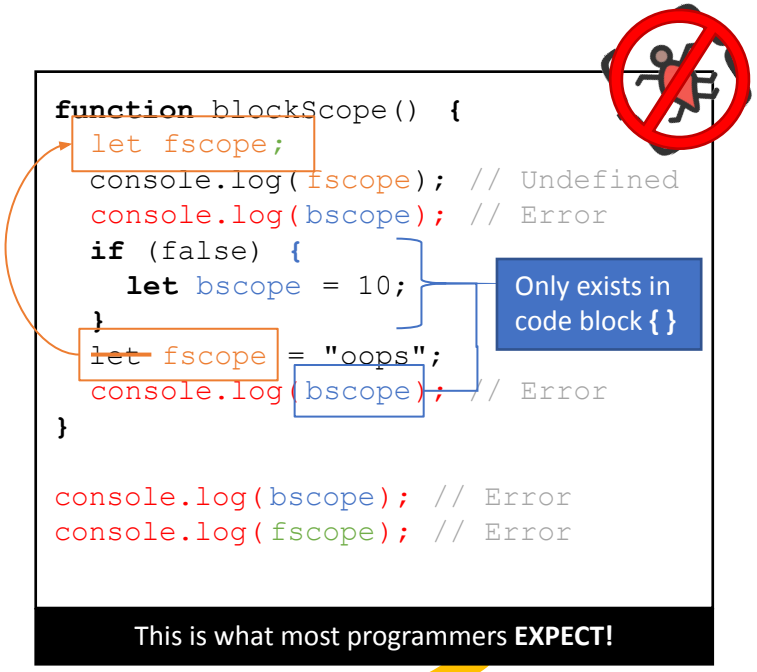
- *var*
  Function scoped. A variable declared outside of a function becomes a global variable.

- *let*
  Code block { } scoped
  This is how most programming languages work with the only difference that variables are "hoisted" to the top of the code block.

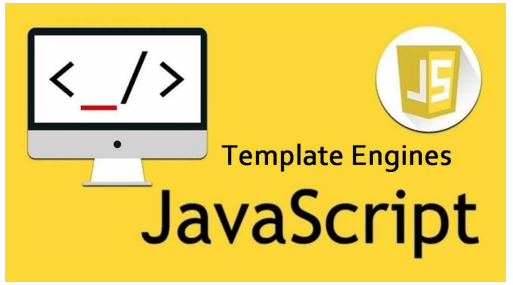
- *const*
  Same scope as `let`, but value cannot be changed after set.

```
const AZ_SALES_TAX = 0.056;
AZ_SALES_TAX = 0; // Error
```





Template Literals
(ECMAScript 6)



- Use ` (~backtick)
  Upper left corner keyboard
- Provides multiline support.

```
let fourscore = "Four score and \n"
+
  "seven years ago our fathers \n" +
  "brought forth, upon this \n" +
  "continent...";
```

```
let fourscore = `Four score and
seven years ago our fathers
brought forth, upon this
continent...`;
```

- Allow for variables / code to be expanded or executed. Use **${ }** wrapper around code or variable.

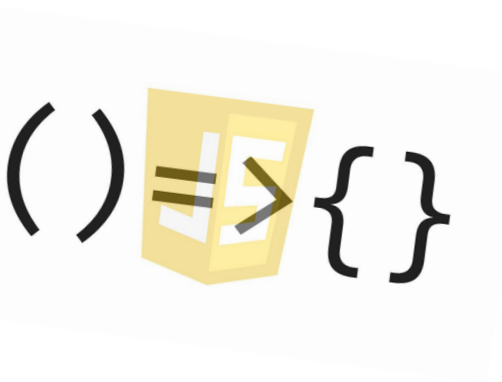
```
let name = "Abby";
let birthdate = new Date(2006, 11, 27); // Dec 27, 2006

console.log("Hi " + name + "! You are " +
((new Date()-birthdate)/(1000*60*60*24)) + " days old today.");
```

```
console.log(`Hi ${ name }! You are
${ ((new Date()-birthdate)/(1000*60*60*24)) } days old today.);
```

Arrow Functions
(ECMAScript 6)

- A function **IS** an object.
  - Assign it to a variable
  - Pass it into a function



```
function isEven(number) {
  return (number % 2) === 0;
};
```

```
isEven(2); // true
isEven(3); // false
```

```
let isEven = function(number) {
  return (number % 2) === 0;
};
```

Anonymous function is assigned to a variable.

Looks like an ARROW!

```
let isEven = (number) => (number % 2) === 0 ;
```

Simplified method

```
let isEven = (number) => {
  // more code
  return (number % 2) === 0;
};
```

Multiple lines can be wrapped with a code block **{ }**. Must explicitly **return** a value.

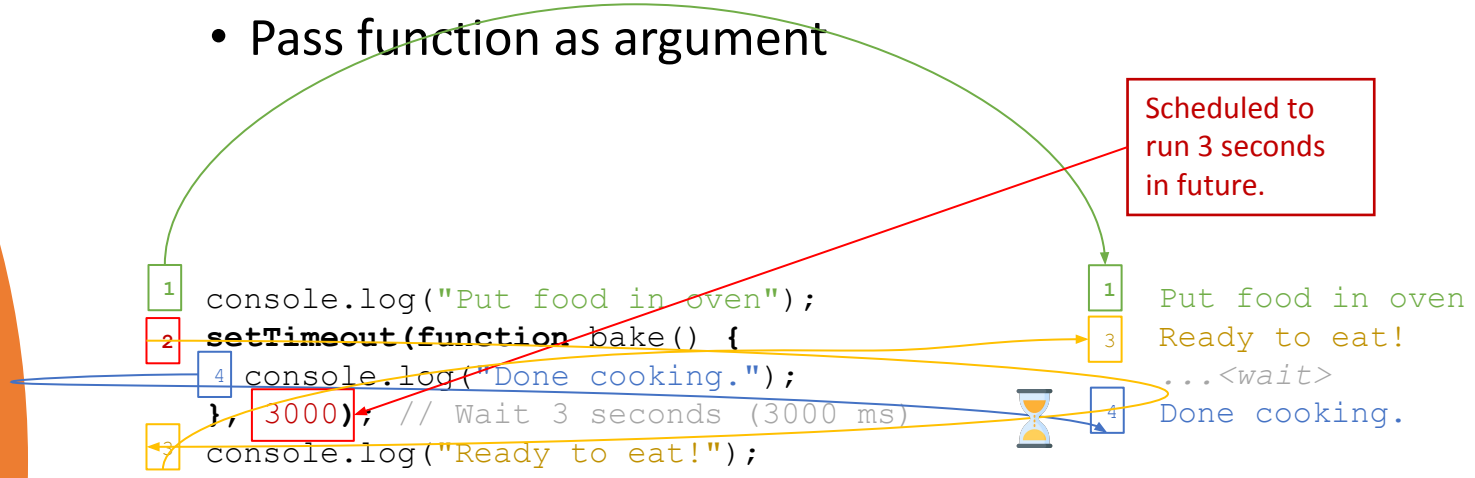
These are all **functionally** equivalent.

DEMO

Scope with Functions

Callbacks

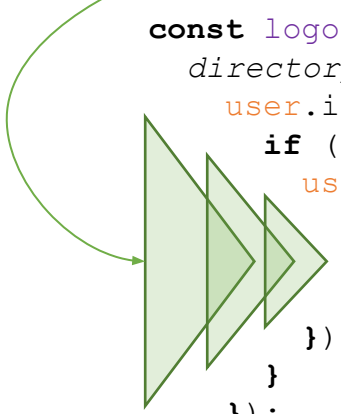
- Asynchronous methods
  - **NOT** sequential
  - Pass function as argument



- Used for blocking operations
  (i.e. something that might not return immediately)
  - Open web page
  - Read from database
- *Don't accidentally invoke method* ***( )***

Promises
(why?)

- Cleaner / simpler version of callbacks
  - Nested callbacks tend to create a "christmas tree".



```
const logon = function(username, password) {
  directory.findUser(username, function(user, err) {
    user.isAuthorized(function(authorized, err) {
      if (!err && authorized) {
        user.validatePassword(password, function(validated, err) {
          if (!err && validated) {
            console.log(`${username} access granted`);
          }
        });
      }
    });
  });
};

logon("bsmith@gmail.com","p@ssW0rd!");
```

Note: Error output / feedback omitted for brevity.

Also sometimes referred to as the pyramid of **DOOM**.

Promises

- An object with the following methods:
  - .**then**() – success
  - .**catch**() – error / something bad happened
  - .**finally**() – success or error (*always*)

```
const logon = (username, password) => {
  directory.findUser(username).then((user) => {
    user.IsAuthorized().then((authorized) => {
      user.validatePassword(password).finally((validated, err) => {
        if (!err && validated) {
          console.log(`${username} access granted`);
        } else {
          console.log(`Incorrect password for ${username}`);
        }
      });
    }).catch((err) => console.log(`${username} not authorized`));
  }).catch((err) => console.log(`${username} not found`););
};

logon("bsmith@gmail.com","p@ssW0rd!");
```

Note: Many JS libraries defined their own implementations / versions or promises. They are not all the same… **CHECK** your documentation!

DEMO

Callbacks & Promises