Bootstrap
- provides pre written CSS classes
- decreases development time
- allows us to more easily incorporate responsive design
- 12 column rows
- available through download, package managers & CDN

JavaScript
- language understood by your browser following the ECMAScript specification
  - traditionally JS was interpreted by your browser but most browsers now will compile it before its run
- ES6 (ES2015)
  additions:
  let/const
  arrow notation
  Symbol
  template literals
- used to interact with our webpage to give it functionality

Incorporating JS into HTML

① Internally
<script> js goes here </script>

② externally
<script src="myscript.js"></script>

put at the bottom of body unless doing how our page is rendered

JavaScript as a Language
- loosely typed
  let x = 'hello';
  x = 25;
  → we don't have to define the type & it can change
- type coercion
  if(5){
  anything can be type coerced to true or false
  }
  "7" + 7 → "77"
  7 + 7 + "7" → "147"
  "7" * 7 → 49
  "cat" * 7 → NaN
  property of type number

Truthy vs Falsy
everything else
0
""
null
undefined
NaN
false

Declaring a variable
var
let
const

Comparison
7 == "7" → allows for type coercion
7 === "7" → equal type + value

Primitive Types
String
number
boolean
undefined
null
Symbol
(everything else is an object)

Let Var & Const

Var
- allows for variable hoisting
- can be redeclared
- can be reassigned ex. var x = true;
- cannot be scoped to a block var x = 250;

let
- allows for reassignment but not redeclaring ex. let x = true; x = 250;
- will not be hoisted
- can be scoped to a block

const
- cannot be reassigned or redeclared
- block scope
- no hoisting

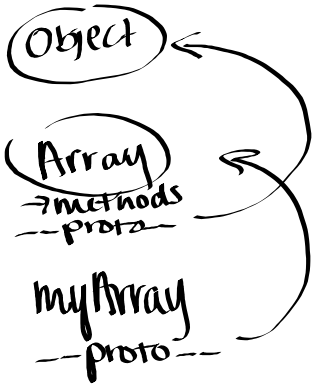
variable Scopes
- global
- lexical scope (function or local scope)
- block scope (only with ES6)

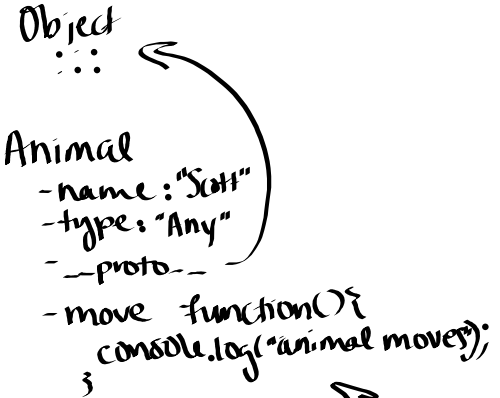
```
var x = true;

if (x) {
    //do something
}

for(var x = 10; x > -1; x--){
    //do something
}
```

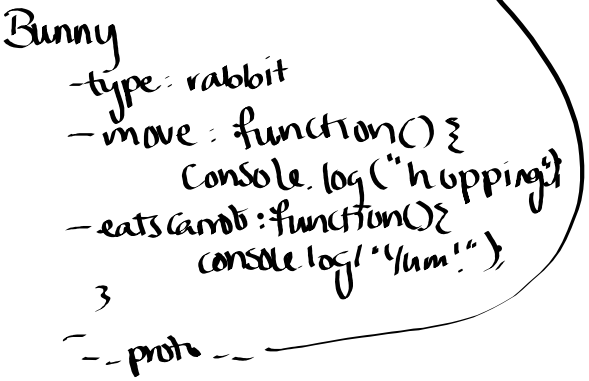
(x is now 0)

```
if (x) {
    //do something else
}
```

Inheritance & the Prototype Chain








```
let bunny = new Bunny();
bunny.type
   → "rabbit"
bunny.move();
   → hopping
bunny.name
   → "Scott"
bunny.runs()
   check up the prototype chain
      for "run" before resulting in an
         error
```

Guard & Default Operators

```
if(str !=null && str.length >5)
```

Logical AND

true & true → 1
true & false → 0
false & true → 0
false & false → 0

Guard Operator

truthy && truthy
truthy && falsy
falsy && truthy
falsy && falsy

"cat" && "dog" → "dog"
0 && 5 → 0
{} && "" → ""

```
let user = loggedIn && username;
```

```
let user = loggedIn? loggedIn:username;
(ternary)
```

Logical Or

true | true → 1
true | false → 1
false | true → 1
false | false → 0

default operator

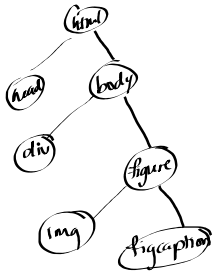
truthy || truthy
truthy || falsy
falsy || truthy
falsy || falsy

```
let currentArray=previousArray|| [];
for(let value of currentArray){
}       //do something
```

DOM - Document Object Model

- → JS representation of our HTML page
- → tree like structure of nodes representing the elements on our page
- → "window" is the global object
- → "document" object allows us to access the nodes in the dom

```
<html>
  <head>
  </head>
  <body>
    <div>
      <p></p>
      <p>...</p>
    </div>
    <figure>
      <img .../>
      <figcaption>:</figcaption>
    </figure>
  </body>
</html>
```



Accessing Elements in the DOM:

→ using document methods

```
document.getElementById(...)
document.getElementsByTagName(...)
document.getElementsByName(...)
<input name="favorite-color" ...>
document.getElementsByClassName(...)
<img class="large-image" src="..." alt="...">
```

return collection of elements

→ node relationships allow us to access related elements

```
[element].children
         .nextSibling
         .previousSibling
         .firstChild
         .lastChild
```

Manipulating Elements in the DOM

```
document.createElement(tagname)
[element].appendChild([element])
[element].removeChild([Element])
```

you can change attributes of your accessed elements as well