Scalar Objects
- String, Number, Boolean, undefined, null
- Assigned by value
  - `let a = 2;`
  - `let b = a;`
  - `let c = 2;`

- Three variables, each containing 2
  - `a === c // true`

Compound Objects
- Array, Object
  - Assigned by reference
    - `let a = {"val": 2};`
    - `let b = a;`
    - `let c = {"val": 2};`

- There are two references (a and b) to one object
  - `a === b // true`
  - `a === c // false`

We’ve been using this
- `let temp = document.getElementById("tempPgh");`
  - `temp.textContent = "72°";`

- Here `temp` is a new reference to an existing object (part of the DOM). Changing its `textContent` property changes the DOM.
  - `// does not change the DOM`
    - `let temp = document.getElementById("tempPgh");`
    - `let tStr = temp.textContent; // a whole new string`
    - `tStr = "72°";`

Functions
- Since functions are objects, they have properties.
  - `let f = function () {
      console.log(f.animal);
  }
  f.animal = "cow";`

- Functions are also assigned by reference
  - `let g = f;`
  - `g.animal = "sheep";`

- There is one function, and it prints “sheep”
- `f` and `g` are both references to the function

Garbage Collection
- Javascript has garbage collection, unlike C, C++ or Java.
- Notice we never allocate space for objects, and we don't have to free them.
- What is garbage collection?
Garbage Collection

- Javascript has garbage collection, unlike C, C++ or Java.
- Notice we never allocate space for objects, and we don't have to free them.
- What is garbage collection?
- The interpreter keeps track of the number of references to each variable. If the number of references goes down to zero, it reclaims the memory, and the variable is gone.
- How do references to a variable disappear?

Variable destruction

- How do references to a variable disappear?
- The reference to a variable is removed when the block it belongs to exits.

```javascript
function changeTemp() {
    let t = document.getElementById("temp");
    t.textContent = "72\u00b0";
}
```
- During the function, the object with id "temp" had two references.
- After the function exits, it has one.

Listener for image download

- In collectPastDoppler, we set a listener for when an image has finished downloading.

```javascript
let newImage = new Image();
newImage.onload = function () {
    addToArray(newImage);
};
```
- This is the fourth listener we've seen (onclick, onload for JSON, setInterval in animation, onload for image)

Callback functions

- The functions called by the listeners — onclick, onload, setInterval — are called callback functions.
- This pattern — set up a listener with a callback function — occurs all over Web code.
- Javascript is designed to handle it gracefully.
- Particular interesting, useful language feature: closure.

Closure

```javascript
function tryToGetImage(dateObj) {
    ...
    let newImage = new Image();
    newImage.onload = function () {
        addToArray(newImage);
    }
    ...
}
```
- The variable newImage belongs to tryToGetImage().
- newImage should disappear when tryToGetImage exits.
- But it is still there when the anonymous callback function runs, much later!

Closure

```javascript
function tryToGetImage(dateObj) {
    ...
    let newImage = new Image();
    newImage.onload = function () {
        addToArray(newImage);
    }
    ...
}
```
- Any function created inside a block creates new references to all the variables from that block.
- This is called a closure.
- We say that newImage is in the closure of the anonymous function.
Closure

- Let's look at a simpler example (from Elequent Javascript):

  ```javascript
  function wrapValue(n) {
    let local = n;
    return () => local;
  }
  let wrap1 = wrapValue(1);
  let wrap2 = wrapValue(2);
  ```

- Type of wrap1 and wrap2?

---

Adding an onclick value

- Consider building the corresponding DOM node in Javascript

  ```javascript
  <div class="bird" id="peacock" onclick="disappear('peacock')">
    
    
  </div>
  ```

- What is the type of birdDiv.onclick?

  STRING, sadly.
How about this?

birdDiv.onclick = disappear('peacock');

- What is the type of `birdDiv.onclick`?

Third try

birdDiv.onclick = disappear;

- What is the type of `birdDiv.onclick`?

Three trys, all wrong...

birdDiv.onclick = "disappear('peacock')";
birdDiv.onclick = disappear('peacock');
birdDiv.onclick = disappear;

- What is the type of `birdDiv.onclick`?

How about this?

birdDiv.onclick = disappear('peacock');

- What is the type of `birdDiv.onclick`?

undefined, since `disappear` is executed on the right-hand side, and it does not have a return value.

Third try

birdDiv.onclick = disappear;

- What is the type of `birdDiv.onclick`?

It's a function, but it's not going to work without it's parameter! It has to know which one to delete!

Do it using a closure

```javascript
function addOnOnclick(element, func, param) {
    function noarg() {
        func(param);
    }
    element.onclick = noarg;
}
addOnOnclick(birdDiv, disappear, "peacock");
```

- Notice we define a function inside another function.
- What is the type of `birdDiv.onclick`?
Do it using a closure

function addOnclick(element, func, param) {
  function noarg() {
    func(param);
  }
  element.onclick = noarg;
}

When does "noarg" get called?

Closure

- The closure of a Javascript function contains all the variables in the scope within which the function was defined.
- The closure is part of the function object.
- The closure of "noarg" is "addOnclick".
- The local variables declared in "addOnclick" are available to "noarg", forever.
- If we call "addOnclick" multiple times, we can declare different instances of the local variables, and versions of "noarg" with different closures.

See example in poultry3.js

- Additional things to notice…
  - Uses DOM methods querySelector() and querySelectorAll(), more general variants of getElementById
  - Function can be called before it is defined, thanks to function hoisting.

Anonymous closure

- Closure and anonymous functions are often combined in a powerful but (initially...) mysterious pattern.

function addOnclick(element, func, param) {
  element.onclick = function() {
    func(param);
  }
}

Do we prefer the anonymous version?