ECS 162
WEB PROGRAMMING

4/24

Green dot under Fairfield

- What is going on?
- What is doppler radar anyway?
- It returns signal when the radar – a bit above ground level – encounters moving objects.
- Usually this is rain, snow, hail, might be bugs or birds.

Green dot under Fairfield

- What is going on?
- What is doppler radar anyway?
- It returns signal when the radar – a bit above ground level – encounters moving objects.
- Usually this is rain, snow, hail, might be bugs or birds.

Rio Vista wind farm

What could go wrong?

```
let weather = {"desc" : "sunny"}
function check(w) {
        if (w.desc = "raining")
                { console.log("umbrella!"); }
        }
check(weather);
console.log(weather.desc);
```

- Trick question – what does this print?

What could go wrong?

```
let weather = {"desc" : "sunny"}
function check(w) {
        if (w.desc = "raining")
                { console.log("umbrella!"); }
        }
check(weather);
console.log(weather.desc);
```

- Trick question – what does this print?

  umbrella! and then raining. Used =, not ==

Side effect of bug in function…

- Could we prevent errors by declaring the object with const, rather than let?
- Sadly no! Const prevents the reference weather from being re-used for some other object. But the object itself is still mutable.
- So const is mostly useful for primitive scalar data types.

Today

- More closure examples
- Closures are what people from industry ask about when they want to know if we are running a serious Web programming course
- In Javascript, closures are the answer to life, the universe and everything…

Closure quiz

```
let x = "outer";

function f() {
   let x = "inner";
   let a = function () {
         console.log(x);
   }
   return a;
}

let a = f(); a();  console.log(x);
```

What does it print?

Closure quiz

```
let x = "outer";

function f() {
   let x = "inner";
   let a = function () {
         console.log(x);
   }
   return a;
}

let a = f(); a(); console.log(x);
```

What does it print?
"inner", then "outer".

The closure of a() contains all the local variables of f(). The local variable x inside of the function hides the global variable.

Global variables

- A variable declared outside of any function is global
- In strict mode:

```
function accidentallyGlobal() {
    accident = 2;
}
accidentallyGlobal();
console.log("accident:", accident);
```

- Runs fine in "sloppy mode", accidentally creates a global variable accident.
- How do you make your programs strict mode?

Global variables

- A variable declared outside of any function is global
- In strict mode:

```
function accidentallyGlobal() {
    accident = 2;
}
accidentallyGlobal();
console.log("accident:", accident);
```

- Runs fine in "sloppy mode", creates a global variable accident.
- How do you make your programs strict mode?

First line of .js file should be "use strict"

Using a global variable

```
let imageArray = []  // global variable to hold stack of images for animation
let count = 0;       // global var

function addToArray(newImage) {
    if (count < 10) {
        newImage.id = "doppler_"+count;
        newImage.style.display = "none";
        imageArray.push(newImage);
        count = count+1;
        if (count == 10) {
            console.log("Got 10 doppler images");
        }
    }
}
```

- Count is global so that it persists between calls to addToArray().

Static variables

- Static variables are local, but persist through multiple calls to the function.
- Javascript does not have them!
- But it is so uncool to use globals instead.
- Why?

Static variables

- Static variables are local, but persist through multiple calls to the function.
- Javascript does not have them!
- But it is so uncool to use globals instead.
- Why?

  Because it often introduces bugs. It is easy to accidentally change a global variable, since it can be changed anywhere in the file.

Function property as static variable

```
function persist() {
    if (persist.x == undefined ) {
        persist.x = 0;
    }
    persist.x++;
    console.log(persist.x);
}
```

- People seem to think this is better, but it isn't

It's still global!

```
function persist() {
    if (persist.x == undefined ) {
        persist.x = 0;
    }
    persist.x++;
    console.log(persist.x);
}

console.log(persist.x);   // Works! Bad! Plus, ugly…
```

Solution using a closure

```
function makeFunctionWithStatic() {
        let count = 0;
        let newFun = function () {
                count = count+1;
                if (count >= 5)
                        {count = 0;}
                console.log(count);
                }
        return newFun;
        }
```

Function that returns a function that has a static variable.

Solution using closure

```
let counter = makeFunctionWithStatic();
for (i=0; i<10; i++)
        {counter();}
```

- Now count is static - it persists between calls to counter() – and also local to counter().

Objects

- A Javascript object is, at heart, a data structure mapping keys to values (map/dictionary/hash table/associative array).
- While this is super-simple and useful, it does not cover some important things:
  - Private data and methods
  - Inheritance
  - Instantiation
- These are also available in Javascript via classes

Public vs private data

let DavisWeather = {"desc": "sunny"};

- Any code with access to weather also sees weather.desc and weather.temp – that is, these properties of the object are public.
- Javascript does not really have private data associated with objects, but we fake it with function scoping.
- In ES6 (the most recent version of Javascript), we do this by declaring a class, which gives us a constructor method.

Class

```
class Weather {
  constructor (desc) {
     this.desc = desc;
   }
}
let DavisWeather = new Weather("sunny");
```

- Defines a class of objects.
- An instance of a Weather object is created using the new keyword.
- The constructor function might take arguments.

Constructor functions

- By convention, the name of a class begins with a capitol letter
- Constructor function parameters control the initial settings of properties
- "this" in the constructor function contains the object being created. As opposed to, say, the class or the constructor function, which are also objects…

Class

```
class Weather {
  constructor (desc) {
     this.desc = desc;
   }
}
let DavisWeather = new Weather("sunny");
```

- So far, the resulting object (DavisWeather) is the same as the version declared with an object literal.

Local variables in constructor

```
class Weather {
  constructor (desc, day) {
     this.desc = desc;
     let _day = day;  // a local variable in a function
     this.report = function () {
          console.log("On ", _day, " the weather is ",desc);
   } } }
```

- The variable _day is included in the closure of the report method.
- Local variables in the closure of a method defined in the constructor are not visible outside the constructor or method.

Constructor with multiple methods

- Add a method:

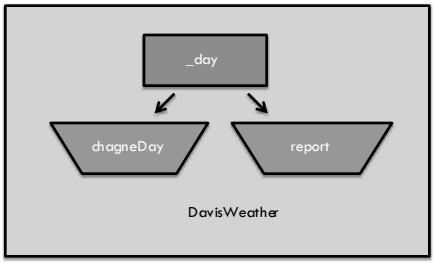
```
this.changeDay = function () {
        _day = "Tuesday";
    }
```

- We see that it changed:

```
DavisWeather.changeDay()
DavisWeather.report()
> On Tuesday the weather is sunny
```

The two functions share the same closure – the local variables of the constructor function.

The closure is shared



Private data

- This is not exactly like private data in a C++ or Java class
- But it servers the same purpose, more or less.
- The local variables of the constructor in the closure are private to the class, but persist throughout the lifetime of the object.