JavaScript (part 2)

CSC309

Mark Kazakevich

- Last time, talked about variables and scope
- Now we'll take those concepts a bit further, and talk more about the nature of JS
  - "Functional" and "Object Oriented" properties of the language
  - How they all come together

Functions in JS are “first-class objects”

- This means they can be:
  - Stored in a variable
  - Passed as an argument to a function
  - Returned from a function

- Essentially used as a value anywhere values are used

‘Anonymous’ functions

- Functions can be passed around without names

- Can call them using **Immediately Invoked Function Expressions**
  - Wrap function in brackets and call `();`

```
(function () {
  console.log(‘anonymous’);
})();
```

Immediately Invoked Function Expressions

```
(function foo() {
 const a = 7;
 console.log(a);
})();

// 7
```

Can give it a name for help in stack trace and self-documentation.

Closures

- JS supports **closures**
  - References to **scopes** that can be passed around

- Allows function/block scopes to be **preserved** even after they finish executing

- Function can “carry baggage” with it from where it was created

Closures

```
function foo() {
   let a = 2;
   function inner() {
      console.log( a ); // 2
   }
   return inner;
}

const bar = foo(); // foo returns
bar();
// 2
```

inner will carry with it all variables in the scope **when it was defined**

In this case, a

```
function foo() {
   let a = 2;
   function inner() {
      console.log( a ); // 2
   }
   a = 5;
   return inner;
}

const bar = foo();
bar();      // 5
```

a can still change in `foo()`, and `inner()` will register those changes in the carried scope until foo() returns.

Closure Demo

Arrays in JS

- You can make an **array** (list) in JS using square brackets
- `const a = [1, 2, “hello”, function() {...}]`
- Indexing: `a[0]`
- Mutable: `a[1] = 73`

Arrays in JS

- How do you find the length?
  - You don't have to use a function

```
a.length // 3
```

- The array has a `length` property attached to it

- Type of a?

```
typeof(a)  // "object" <- not a primitive type
```

JS Objects

Objects

- An object in JS is simply a set of key-value pairs
- Keys are called “properties”
  - Can be strings (or Symbols in ES6)
- Values can be of any type
  - Can make complex data structures

Objects creation

- You can create object literals:

```
const student = { name: ‘Jimmy’, year: 2};
const student = {“name”: ‘Jimmy’, “year”: 2};
```

  - Quotes are optional

- Object properties retrieved by `student.name` or `student[“name”]`

Objects properties

- Properties can be added and changed

```
> student.year = 3
> student.age = 20
> student
{name: "Jimmy", year: 3, age: 20}
```

We used **const** to declare `student`, but can still modify its properties - we just can't re-assign `student` directly.

Functions as properties

- Since functions can be stored as values, we can put them into objects

```
> student.sayName = function () {
        console.log(‘My name is ’ + this.name);
    }
> student.sayName()
‘My name is Jimmy’
```

What is “this”?

Demo

`this`

- Refers to the containing object of the **call-site** of a function, not where the function is defined.

- Context-dependent
  - Value of `this` is not obvious from reading function definition

- Can be changed by using `bind()`, `call()`, `apply()`