Week 5 Lecture 14

Theory

What’s in this lecture?

- Intro to Object-Oriented Programming in JavaScript

Recap: FP

- In Functional Programming (FP), strive to separate programs into mostly *pure* and fewer impure functions
- *pure* functions compute results based solely on their inputs: that is, they have no side-effects
- Impure functions may cause side effects, such as assigning to a global variable, logging to console, or calling alert()

Object-Oriented Programming

- In Object-Oriented Programming (OOP), *objects* and *classes* are used to organize programs
- A *class* is a template for creating new *instances* of objects
- For example, a Person class that can create two (or more) instances, "Tom" and "Jerry"
- The Person *class* contains functions that know how to work with Person instances

Object-Oriented Programming

- Object-Oriented Programming strives to encapsulate *private* (or internal) object state and functionality from *public* (or external) state and functionality
- Careful choice of where to place functions and data is the art of program design
- For example, splitting up models and validation from processing and control from presentation logic

OOP Person

```
// used to initialize a person instance data
function Person(name, age) {
  this.name = name;
  this.age = age;
}

// functions for Person instances
Person.prototype = {
  // define functions here...
};
```

OOP Person

```
Person.prototype = {
  greet : function() {
    return “Hi, I’m “ + this.name;
  },
  isOlderThan : function(other) {
    return other.age < this.age;
  }
};
```

Working with Person

```
var tom = new Person("Tom", 21);
var jerry = new Person("Jerry", 64);

console.log(tom.greet());
console.log(tom.isOlderThan(jerry));
```

Using Object.create

- Object.create is used for initializing objects in a cleaner way than new
- Please read prototypical inheritance for a great tutorial
- Can set "enumerable : true" for properties that should be public (for example, a "value" property)
- Can leave "enumerable : false" for properties that should not be exposed (for example, references in data structures)

OOP List

```
function ListNode(value, next) {
  this.value = value;
  this.next = next;
}

ListNode.prototype = {
  head : function() { return this.value; },
  tail : function() { ... }
};
```

OOP List

```
function List() {
  head : null,
  size : 0
}

List.prototype = {
  insert : function(val) { ... },
  remove : function(val) { ... },
  contains : function(val) { ... },
  isEmpty : function() { ... }
};
```

OOP vs. FP

- Compare the FP:
  var next = list_head(data);
- With OOP:
  var next = list.head();
- In FP, we create pure functions that know how to operate on data
- In OOP, we group functions with data as an organizational mechanism
- We may combine FP with OOP when class functions do not change state (that is, do not cause side effects by assigning to instance data)

Exercises

- Finish the Object-Oriented implementation of List
- Create Object-Oriented versions of the Doubly-Linked List and Binary Search Tree data structures
- *Without* using inheritance (don't worry if you don't know what this is), create an object model to simulate a parking lot, including ParkingLot, ParkingSpace, Vehicle, Motorcycle, SemiTrailer and Car classes; the ParkingLot instance should support a parkVehicle(vehicle) method, refuse vehicles that don't fit, and be able to say how many spaces are left