1. Section Overview

* Programming paradigms:
  + Functional programming.
  + OOP.



3. OOP Introduction

* In OOP we need:
  + Data/State.
    - Data allows us to keep track of the object.
  + Method.
    - Methods allows us to manipulate the state of the object.
* In OOP we have two types:
  + Class based.
  + Prototype based.
* OOP is all about modeling real world objects and relationships.

4. OOP1 Factory Functions

* In ES6, if the property/key and value are same in an object we just keep only one.
* Factory function: Function that creates objects for us.
* Drawback of factory functions is that it can have same method stored different places in memory.

5. OOP2 Object.create()

* One way is that we can refactor common methods manually, but it won’t know who is calling it.
* To solve this, we can use Object.create to create a prototype chain between two functions.
* Object.create is clean but many has objection over this, so it’s not present in many code bases.

6. OOP3 Constructor Functions

* To avoid previous problem, we use constructor function.
* We only construct the function but don’t return anything.
* To use this function, we need new keyword. (btw many don’t like new)
* new keyword automatically returns an object.
* Any function that is invoked using new keyword is called constructor function.
* As a rule, we need to capitalize the constructor function.
* We can also make constructor function using Function().
* new keyword makes this to point to the object that we created.
* In any regular function the prototype property is useless except the constructor function.
* Built in functions/objects are constructor function.
  + Function() has prototype: {call(), apply(), bind()}
* We can make our own prototype property for a constructor function.
* Constructor function automatically creates prototype chain.
* Arrow function will not work because it is lexically scoped so this keyword can’t find what we are looking for, instead it points to the global object.
* Regular functions are dynamically scoped.

7. More Constructor Functions

* Only way to add properties to the object we have to use this keyword.
* obj.prototype() was created under the hood by new keyword.
* Every function that we create has this and prototype property. Only constructor function has its use.
* Only functions have access to prototype.

Gotcha:

* Function inside of a method, this is not assigned to the object that we want instead it points to window object.
  + To solve this, we can use bind().
  + Or we can reference this before the function, that way the function has the this.

The cons:

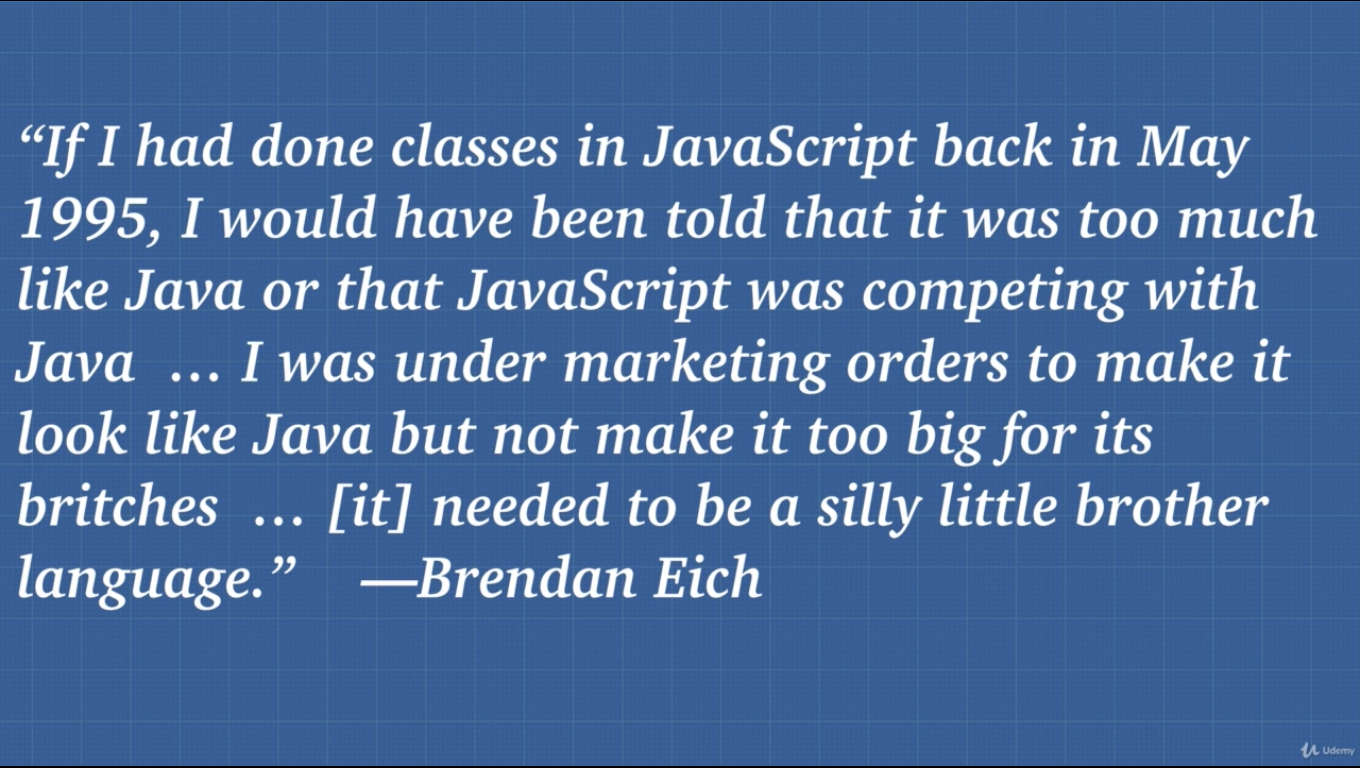
* Prototype is kind of wired, confusing, hard to understand.
  + Using prototype is not much used. Older code bases have used this.
  + This approach is not pretty.
  + OOP is all about the idea of classes.
  + We have to remember to use new keyword, that’s the reason Object.create was created.

8. Funny Thing About JS...

* In JS when we assign a variable, internally it constructs Number() , that’s why we have toString() method.
* JS automatically assumes that we want to use objects instead of primitive.
* JS has constructor functions for everything except null and undefined, to use different method s

9. OOP4 ES6 Classes

* class is new in ES6.
* The constructor gets run when we instantiate or use the new keyword on a class.
* We want to keep all our properties, methods, actions inside of this class an object.
* OOP is modeling real world scenarios, here class is a blueprint.
* Instance happens when we create an object out of any class.
* With the new keyword it’s called instantiation.
* This is not rally OOP yet, class is just syntactical sugar, underneath the hood JS uses the new keyword with the prototype.
* class keyword is still just prototypal inheritance.
* Some call its pseudo classical inheritance because it’s not real classical inheritance.
* We don’t have methods inside of the constructor, it will have memory space every time we create a new instance.



10. Object.create() vs Class

* There is a debate in the JS community, they don’t want to use new or this, because it causes too much confusion.
* Some say that Object.create() is the pure way of doing it instead of pretending that we have class.
* Most of the community don’t use Object.create() especially the new code bases they use class.
* Beauty of JS is it’s multi paradigm.

11. this - 4 Ways

Ways to manipulate this keyword:

1. New binding:
   * The new binding is new way to manipulate this keyword
   * New binding allow us to assign this to the object that we instantiate.
2. Implicit binding (most common):
   * Inside of an object this refers to that object itself.
3. Explicit binding:
   * We dictate exactly what this keyword should be.
   * We need to use bind(), call(), and apply().
4. Arrow function:
   * Gotcha: Function inside of a method is problematic for this keyword, but arrow function can help to determine the object not the window object.

12. Inheritance

* We can copy an object using {...} spread operator, but this can’t copy all the functionality (limited). They do not point to the same place in memory.
* To inherit fully we can use sub class.
  + Inherit form the super class. Class Subclass extends Superclass { }
* To have new functionalities we need to use constructor(param) and super(param). We can use this in this way.

13. Inheritance 2

* When we crated a method for extended class, we made it with the prototype, well JS did it for us underneath the hood.
* Checking the connection of classes with .prototype is very confusing.
* To check connection, we can use instanceof.
* Instance is essentially creating a version of a class.
* Inheritance is inheriting from a higher class.
  + In JS inheritance doesn’t copy all the functionality.
  + It simply creates a connection so no memory loss.
* In JS there are no technical classes. It’s just objects inheriting objects.
  + JS just references object to another object, bit of efficiency.

14. Public vs Private

* In many OOP languages private and public fields are very important.
* In JS we don’t have this.
* We have # sign to make any variable private.
* JS doesn’t have this private or public fields as for now but work in progress.
* The properties and methods are available not private at least for now.

15. OOP in React.js

* Some gained knowledge that we have can be spotted in many OOP examples.

16. 4 Pillars of OOP

1. Encapsulation
   1. OOP puts things inside of an object.
   2. We wrap code into boxes that are related to one another.
   3. These boxes can interact with each other using methods and properties that we make available.
   4. Easy to maintain and reusable (class, packages).
2. Abstraction
   1. It is hiding the complexity from the user.
   2. We use all the methods and properties and no need to worry about all that happens behind the scene.
   3. Idea of abstraction also helps when we use private variables and methods which is in progress for JS.
3. Inheritance
   1. We don’t write the same code we just inherit.
   2. Save memory having shared methods.
4. Polymorphism (many forms)
   1. Definition is heavily debated
   2. Ability to call the same method on different objects and each object responding in different way.
   3. It has the ability to process objects differently depending on their data type or class.
   4. JS is dynamically typed language it limit the amount of polymorphism we can have.
   5. We don’t copy code we reuse functionality from superclass.

* Clear and understandable.
* Easy to extend.
* Easy to maintain.
  + No need to worry about prototype.
  + Inherits from a super class. If modification is required, we just do it in one location.
* Memory efficient.
* DRY.