W04- Object Oriented Programming

Separate code into objects
- constructors - classes - Prototypes - public & Private
- inheritance - objects from objects - Mixins - Chaining function
- this - prototypes/borrowing

Encapsulation - inner workings kept hidden - only essential functionality is exposed.

Polymorphism - Same processes can can be used for different objects

Inheritance - inherit all properties of an object then add more for a new object

Classes - Prototype-based (JavaScript) - build an object, then use that as the basis for upgrading object

Constructor Functions - function that defines the properties and methods of an object

-this - used to represent the object

instanceof - boolean returns true if object is instance of a constructor function

Built-in Constructor Functions - Object, Array, Function

literal syntax : const literalObject = { };

constructedObject = new Object();

Const literalArray = [1, 2, 3];

ConstructedArray = new Array (1, 2, 3);

a single argument =new Array (3); creates an array with length of 3 Undefined.

<< [undefined, undefined, undefined]

W04- Object Oriented Programming

Class Declarations • introduced in ES6

```
class Dice {
    constructor (sides=6) {
      this.sides = sides;
    }

  roll() { ... }
```

ES6 class declarations are preferable
• more succinct • easier to read • implicitly in strict mode

The Constructor Property - All objects have a constructor property
• Can use the constructor property to instantiate a copy of an object

if we want another copy of the redDice object, but if the construct
is unknown:

```
const greenDice = new redDice.constructor(10);
greenDice instanceOf Dice // true
```

Static Methods - static keyword
Methods only accessible from the class, not instances of the cla

Prototypal Inheritance - every class has a prototype property shared by every instance of the class.

```
turtle.prototype.attack = function ( ) { ... something ... }
```

ADDS A FUNCTION THAT IS NOW ACCESSIBLE TO ALL INSTANC
OF TURTLE

FINDING THE PROTOTYPE - .getPrototypeOf( );

ralp.constructor.prototype;

isPrototypeOf( ) — boolean to check if prototype of an instance.

W04 - Object Oriented Programming

Own Properties and Prototype Properties

ralph.hasOwnProperty('name'); //True
ralph.hasOwnProperty('weapon'); //False

Prototype properties and the value is shared with every instance

The Prototype is Live! - if a new property or method is added to the prototype, any instance of the class will inherit them automatically - even if that instance is already created

Overwriting Prototype Properties - an object instance can overwrite any properties or methods inherited from its prototype.

leo.weapon = 'Katana Blades'; // leonardo's weapon has been up becoming an Own property

- own properties take precedence over prototype propertie

What Is the Prototype Used For? - add new properties and methods after a class has been declared.
Should be used for properties that will be same for every instance.

- Class declaration deals with initialization, shared properties and methods
- Any extra methods and properties that need to augment the class after it has been declared can be added using prototype.
- Add any properties or methods that are individual using assignment operator

Public and Private Methods - by default Public

Private properties _color

```
let _color = color;
this.setColor = color => { return _color= c
this.getColor = () => _color;
```

getters and setters provide controlled access.

W04 - Object Oriented Programming

Inheritance - examples of inheritance in discussion of prototype

THE OBJECT CONSTRUCTOR

Object.prototype includes a large number of inherited methods

propertyIsEnumerable() - check if a property is enumerable

Inheritance Using extends keyword

Polymorphism - different Objects can have same method implemented in a different way.

toString() method is inherited from Object.prototype

ADDING METHODS TO BUILT-IN OBJECTS (monkey-patching)

```
Number.prototype.isEven = () => this % 2 === 0;
Array.prototype.first = function() {
    return this[0]; }
Array.prototype.last = function() {
    return this[this.length - 1];
```

• JS community currently frowns on monkey patching

Property attributes and descriptors

value - value of property, undefined by default

writable - boolean expressing whether a property can be change, false by default.

enumerable - boolean expressing whether a property will show when the object is displayed in a for in loop, false by default

configurable - boolean expressing whether you can delete a property or change any of its attributes, false by default.

when assign a value, these all set to true when assignment made

W04- Object Oriented Programming

can see property descriptors:

```
Object.getOwnPropertyDescriptor(me,'name');
  -returns-
    {value:'DAZ',
     writable: true,
     enumerable: true,
     configurable: true}
```

Instead of assignment use defineProperty() method to add properties to an object. This allows each attribute to be set.

```
Object.defineProperty(me,'eyeColor',{ value: 'blue', writable: false,
                      enumerable: true});
```

↑created a property 'eyeColor' that is read only.

GETTERS AND SETTERS

get() and set() methods

me object has age and retirementAge properties. can create a yearsToRetirement property with a get() and set() method

```
me.age = 21;
me.retirementAge = 65;

Object.defineProperty(me,'YearsToRetire',{
  get(){
    if (this.age > this.retirementAge) { return 0;}
       else { return this.retirementAge - this.age;}
  set(value) {
    this.age = this.retirementAge - value;
    return value;
  }
```

Allows getting years to retirement Age based on me.age and me.retirementAge.

Allows setting me.age based on value and me.retirementAge

W04 - Object-Oriented Programming

CREATING OBJECTS FROM OTHER OBJECTS

avoid using classes through creating new objects based on another object acting as a blueprint or prototype

```
const Human = {
  arms: 2,
  leges: 2,
  walk() { console.log('walking'); }
}
```

```
const lois = Object.create(Human);
```

^ a new object that inherited all the properties of Human

```
const jimmy = Object.create(Human,
  { name: {value: 'Jimmy Olsen', enumerable: tr
    Job: {value: 'Photographer', enumerable: true
  });
```

^ a new object that inherited properties from Human and added additional properties

OBJECT BASE INHERITANCE - Super-class, becoming the prototype of other objects

```
~~const~~ Superhuman = Object.create(Human);

Superhuman.change = function() {
  return `${this.realName} goes into a phone box and comes out
    as ${this.name
```

init() method

```
Superhuman.init = function (name, realName) {
    this.name = name;
    this.realName = realName;
    this.init = undefined; // makes it only possible to call method once
    retun this;
  }
```

Object Prototype Chain - Creating Objects from Objects creates a prototype chain

```
Human.isPrototypeOf(Superhuman); // true
Superhuman.isPrototypeOf(batman); // true
```

Mixins - a way to add properties and methods of some objects to another object without using inheritance

```
Object.assign(); // copied by reference
```

W04 - Object-Oriented Programming

Chaining Functions - if a method returns this, it can be chained together to form a sequence

Binding this - this points to the object calling the method but can lose scope, pointing to the global object instead

USE that = this; Before the nesting

```
superman.findFriends = function(){
  const that = this;
  this.friends.forEach(function(friend){
     console.log(`${friend.name} is friends with ${that.name}`);
  });}
```

prevents losing scope in nested function

USE bind(this) - bind() is a method for all functions to set the value of this in the function while it is still in scope

```
superman.findFriends = function(){
  this.friends.forEach (function (friend){
     console.log (`${friend.name} is friends with ${this.name}`);
  }.bind(this);
}
```

USE for of INSTEAD OF forEach() - doesn't require a nested function
USE ARROW FUNCTIONS

BORROWING METHODS FROM PROTOTYPES - can borrow without inheriting all properties and methods

```
const fly = superman.fly; // created a fly() function by reference
fly.call(batman); // call method on another object
```

BORROWING ARRAY METHODS

```
const slice = Array.prototype.slice; // create slice() function by reference
slice.call(arguments, 1,3); // call on
```

W04 - Object-Oriented Programming

COMPOSITION OVER INHERITANCE - Some problems associated with inheritance

Advocate creating small objects describing single tasks or behaviors, using those as building blocks for more complex objects

- Like Java ?

Make classes 'skinny', few properties and methods
Keep inheritance chains short.