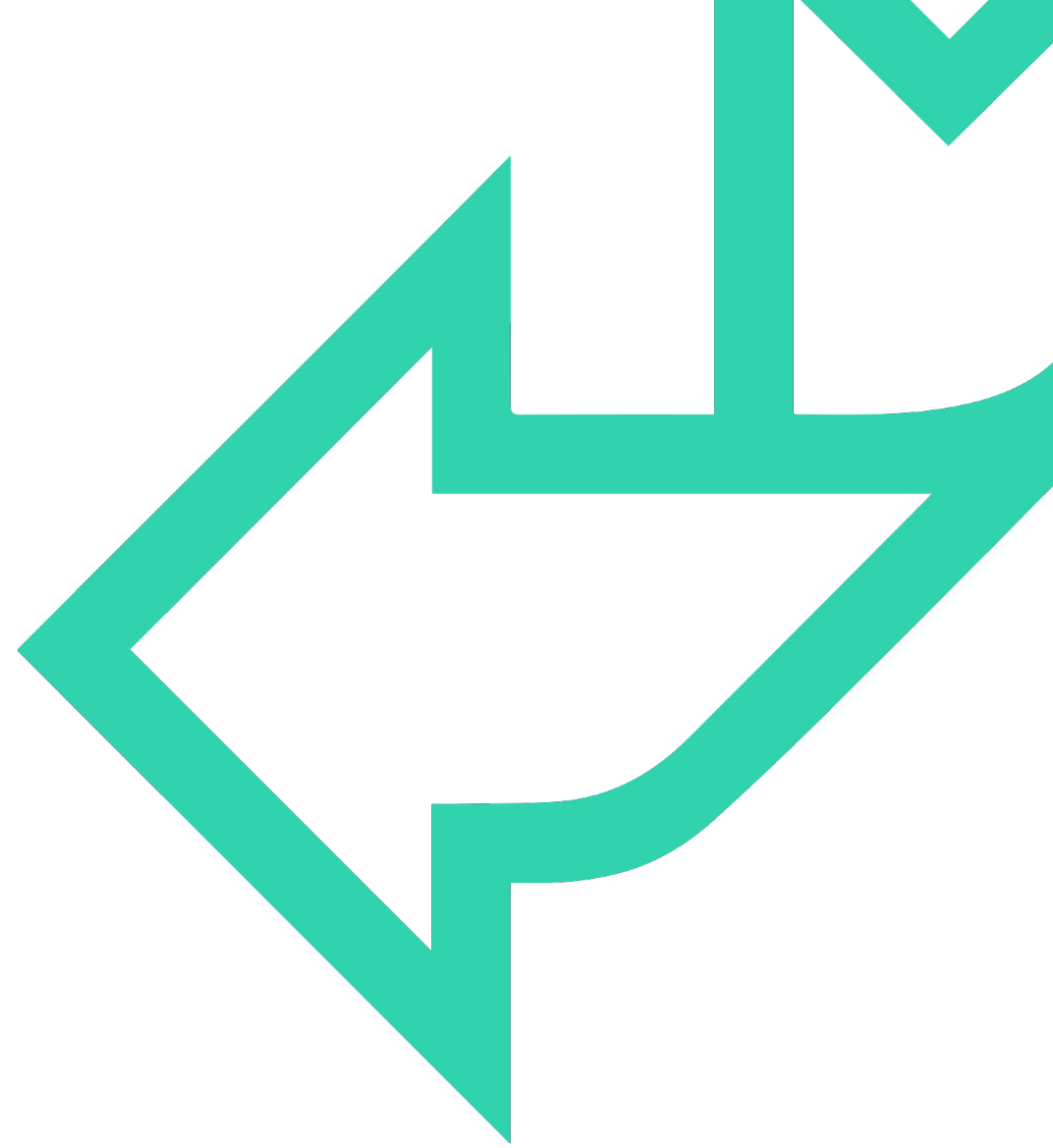




Object Orientated JavaScript

JavaScript Fundamentals





Introduction

Objects revisited

- Object notation
- Scope

Creating your own objects

- Adding functions to objects
- Constructors
- Prototypes
- Chaining objects

Sealing objects

- Defend against unexpected object mutation



QA JavaScript objects (1)

- Everything in JavaScript is an object
 - Functions, dates, DOM elements
 - How we extend the language with our own types
- Creating...
 - Use **new** keyword or { }
- Properties
 - Use *dot notation* or *object literal* notation

```
let myBike = new Object(); // using new
myBike.make = "Honda";
myBike.model = "Fireblade";

let myBike2 = {                // using {}
  make: "Honda",
  model: "Fireblade"
};

myBike.make = "Yamaha"; // Dot
myBike["model"] = "R1"; // Obj Lit
```

QA JavaScript objects (2)

- Use `for...in` loop to iterate over the properties

```
let myBike = {  
  make: "Honda",  
  model: "Fireblade",  
  year: 2008,  
  mileage: 12500,  
}  
  
for (let propName in myBike) {  
  print `${propName} :: ${myBike[propName]}`;  
}
```

QA Classes

```
class Car {  
  constructor (wheels, power) {  
    this._wheels = wheels;  
    this._power = power;  
    this._speed = 0;  
  }  
  
  accelerate(time) {  
    this._speed = this._speed + 0.5*this._power*time;  
  }  
}  
const myCar = new Car(4, 20); //constructor called
```

- Syntactic sugar over prototypal inheritance
- Gotcha: NOT hoisted like functions
- Executed in strict mode
- Private properties are prefixed with an underscore
 - Purely convention as there is no notion of private scope for properties in JavaScript



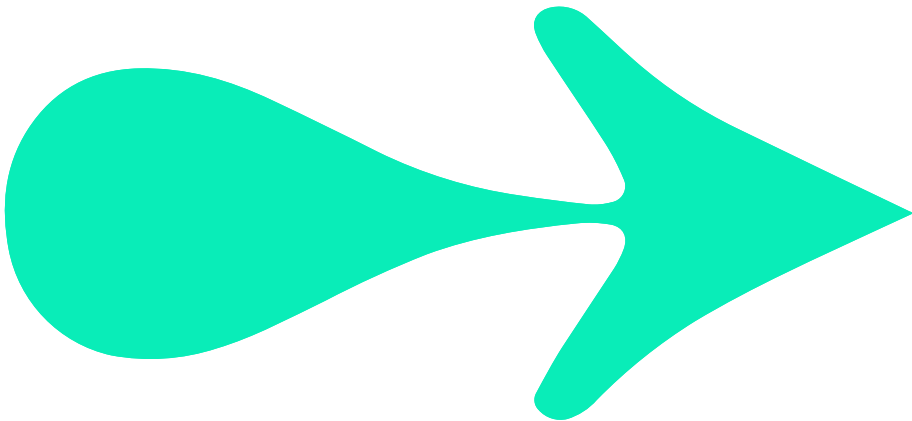
Accessing properties


Object Oriented Programming has a concept called Encapsulation.

- This means 'private' properties should not be accessible directly
- Should use 'accessor' or 'getter' methods to retrieve the value
- Should use 'mutator' or 'setter' methods to change the value
 - Allows the class to decide if the value is permissible

Two ways to achieve in JavaScript:

- Write a method called `getPropertyName()` that returns the value of the property
- If value can be changed, write a method called `setPropertyName()` with logic to change the value
- Use `get` and `set` keywords instead of these functions
 - GOTCHA: cannot use the property name as the 'name' of the function





QuickLab 14a – Creating a class

- Create a class and some methods
- Instantiate the class and use its methods

QA Classes: extends to inherit

- The extends and super keywords allow sub-classing (i.e., inheritance)

```
class Vehicle {
    constructor (wheels, power) {
        this._wheels = wheels;
        this._power = power;
        this._speed = 0;
    }

    accelerate(time) {
        this._speed = this._speed + 0.5*this._power*time;
    }
}

class Car extends Vehicle {
    constructor (wheels, power) {
        super(wheels, power); //call parent constructor
        this._gps = true;      //GPS as standard
    }
}

const myCar = new Car(4, 20);
```


QA Inheritance in action – custom error handler

- JavaScript has inbuilt Error object (along with many other inbuilt objects)
 - Through inheritance, we can create our own error types

```
function DivisionByZeroError(message) {  
    this.name = "DivisionByZeroError";  
    this.message = (message || "");  
}  
  
DivisionByZeroError.prototype = new Error();
```

QA Classes: static

- The **static** keyword allows for method calls to a **class** that hasn't been instantiated
- Calls to a **static** function of an instantiated class will throw an error

```
class Circle {  
    constructor (radius, centre) {  
        this.radius = radius;  
        this.centre = centre;  
    }  
  
    static area(circle) {  
        return Math.PI * Math.pow(circle.radius,2);  
    }  
}  
  
const MY_CIRCLE = new Circle(5,[0,0]);  
console.log(Circle.area(myCircle)); //78.53981633974483
```

QA Sealing objects to prevent expando errors

- Extensibility of objects can be toggled
- Turning off extensibility prevents new properties changing the object
 - `Object.preventExtensions(obj)`
 - `Object.isExtensible(obj)`

```
let obj = {  
    name: "Dave";  
};  
print(obj.name); //Dave  
  
console.log(Object.isExtensible(obj));  
// true  
  
Object.preventExtensions(obj);  
  
obj.url = "http://ejohn.org/";  
//Exception in strict mode  
//(silent fail otherwise)  
  
console.log(Object.isExtensible(obj));  
//false
```



Review



Objects revisited

- Object notation
- Scope

Creating your own objects

- Adding functions to objects
- Constructors
- Prototypes
- Chaining objects

Sealing objects

- Defend against unexpected object mutation



QuickLab 14b – extend the class

- Extend the class made in 15a
- Add new properties to extended classes
- Override methods
- Use class instances