Objects & OOPS in Javascript

Object properties
Object Literals
OOPS in Javascript
Constructor Functions
Inheritance

Objects in Javascript

- The simple types of JavaScript are numbers, strings, booleans (true and false), null, and undefined. All other values are objects.
- Objects in JavaScript are mutable keyed collections. In JavaScript, arrays are objects, functions are objects, regular expressions are objects, and, of course, objects are objects.
- An object is a container of properties, where a property has a name and a value. A property name can be any string, including the empty string.
- Property value can be static or dynamic which are functions(methods)
- Example {Car_Obj —> name: Honda, year: 2018, age: currentyr-2018}

Object Literals

- Object literals provide a convenient notation for creating new object values.

```
var flight = {
    airline: "Oceanic",
    number: 815,
    departure: {
        IATA: "SYD",
        time: "2004-09-22 14:55",
        city: "Sydney"
}, arrival: {
        IATA: "LAX",
        time: "2004-09-23 10:42",
        city: "Los Angeles"
}, name_number: function(){return this.airline +
this.number}
};
```

Object Literals

- Updating property value can be done with simple assignment
- A property's name can be any string, including the empty string. The quotes around a property's name in an object literal are optional if the name would be a legal JavaScript name and not a reserved word.
- dot operator can be used to retrieve properties
- “undefined” is produced when property doesn’t exist
- "

OOPs/OOJs in Javascript

- JavaScript has a class-free object system[classes introduced in ECMA 2015] in which objects inherit properties directly from other objects.

- Every object is linked to a prototype object from which it can inherit properties. All objects created from object literals are linked to Object.prototype, an object that comes standard with JavaScript.

- Functions in JavaScript are objects. Thus they can have methods

Constructor Function = Class

JavaScript uses special functions called **constructor functions** to define and initialise objects and their features.

```
function Person(name) {
   this.name=name;
   this.greeting = function() {
     alert('Hi! I\'m ' + this.name + '.');
   };
}
Person1 = new Person(“john”);
```

Classes and Inheritance

“extends” keyword can help us create a sub-class of a parent class.

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  present() {
    return 'I have a ' + this.carname;
  }
}

class Model extends Car {
  constructor(brand, mod) {
    super(brand);
    this.model = mod;
  }
  show() {
    return this.present() + ', it is a ' + this.model;
  }
}
```

Problem

```
function Shape(name, sides, sideLength) {
  this.name = name;
  this.sides = sides;
  this.sideLength = sideLength;
}
```

//
Add a new method to the Shape class's prototype, calcPerimeter(), which calculates its perimeter (the length of the shape's outer edge) and logs the result to the console.

Create a new instance of the Shape class called square. Give it a name of square and a sideLength of 5.

Call your calcPerimeter()