Friday, 8 March 2019

Scooping

This small portion of code is to show the life of a variable (and functions);

```
thisShouldWork();//defined down
console.log("typeof butNotThis: " + typeof butNotThis);

if (true) {
    var doIExist = variableLife;
}

doIExist();

var x = function howAboutMe() {
    console.log("sorry to disappoint; x() call is the one that worked");
};

console.log("typeof howAboutMe: " + typeof howAboutMe);
x();

// ========================================================

function thisShouldWork() {
    console.log("functions are hoisted");
    //which means you can call it and define it later
}
```

```
var butNotThis = function() {
    console.log("but variables aren't");
};

function variableLife() {
    console.log("\n");
    console.log("variables exist from when they are created");
    console.log("to when the function terminates");
    console.log("\n");
}
```

JS modules

To make it easy to maintain and better reuse our code, we need to use modules; these are just files or group of files that contain some chunk of code solving the same problem. It could be a library.

e.g Let's create a library that has a sum and subtract functions

```
//smallLib.js
function sum(x,y){
  return x+y;
}

function sub(x, y){
  if(x>y){
    return x-y
  }
  return y-x
}
//to make our smallLib file availble to other files we use exports keyword
module.exports = {sum, sub} //exporting the an object with the 2 funcs
```

```
//calc.js
var sum = require('./smallLib').sum // we use require to import the file
like we use #include in C
var sub = require('./smallLib').sub // in Advanced JS we can use import

var a1 = Math.random()*1000
```

```
var a2 = Math.random()*1000

var res = sum(a1, a2)
console.log(res)
```

Classes in Js

Classes in JS can be thought as a better way to deal with big objects or like special functions.

```
//let's say we have an object called Car
var Car = {
  tyres: 4,
  Windows: 6,
  turnLeft: function() {console.log('turning left ...')},
  turnRight: function() {console.log('turning right ...')}
}

//calling the object property (tyres or windows in this case)
console.log(Car.tyres) // 4
//calling the object method
Car.turnRight // turning right ...
//_____________________________________________________________________
function Car(){
  this.tyres = 4 // this here refers to the function itself
  this.windows = 6
  function turnLeft(){
    console.log('turning left ...')
  }
  function turnRight() {console.log('turning right ...')}
}
var myCar = new Car() // creating a new object of type Car
console.log(myCar.tyres) // 4

//But sometimes the methods can be complex and long; a class can deal with
//it in a better way.
```

We use the “**class**” keyword to create a class

```
class Car {
   constructor(){
       this.tyres = 4
       this.windows = 6
     }

     turnLeft(){
       console.log('turning left ...')
     }
    turnRight(){
      console.log('turning right ...')
    }
  }

  var myCar = new Car() //here, we're creating a new instance of the class
//Car
  console.log('My car has '+myCar.tyres+' tyres')
//My car has 4 tyres
```

A **constructor** is a special method that helps to initialize or create properties that belong to the class.
We can also pass parameters as we create a new instance of the class. The parameters will be passed to the constructor of the class.

```
class Car {
  constructor(type, windows=6){//setting 6 as the default number of windows
    this.tyres = 4
    this.windows = windows
    this.type = type
  }

  turnLeft(){
    console.log('turning left ...')
  }
  turnRight(){
    console.log('turning right ...')
  }
```

```
  getCarType(){
    return 'I own a very nice '+this.type+'.'
  }
}

var myCar = new Car('sedan')
console.log('My car has '+myCar.tyres+' tyres'+ ' and '+myCar.windows+'
windows.')
console.log(myCar.getCarType())

var myCar = new Car('convertible', 1)
console.log('My car has '+myCar.tyres+' tyres'+ ' and '+ myCar.windows+'
windows.')
console.log(myCar.getCarType())
//My car has 4 tyres and 6 windows.
//I own a very nice sedan.
//My car has 4 tyres and 1 windows.
//I own a very nice convertible.
```

Inheritance with JS classes: You can create a class that uses some of the properties or methods of another class (this one will be called a super class or the parent class).
To access the constructor of the super class we will use the **super** keyword .
E.g: We want to create a class that defines **Tesla** car , it's an electric car which we will take as the only thing that differentiates it from other cars here.

```
class Car {
  constructor(type, windows=6){//setting 6 as the default number of windows
    this.tyres = 4
    this.windows = windows
    this.type = type
  }

  turnLeft(){
    console.log('turning left ...')
  }
  turnRight(){
    console.log('turning right ...')
  }
```

```
  getCarType(){
    return 'I own a very nice '+this.type+'.'
  }
}

class Tesla extends Car {
  constructor(type, windows){
    super(type, windows)
    this.electricCar = true
  }

}

var myCar = new Tesla('sedan')
console.log('My car has '+myCar.tyres+' tyres'+ ' and '+myCar.windows+'
windows.')
console.log(myCar.getCarType())
console.log(myCar.electricCar)

var myCar = new Tesla('convertible', 1)
console.log('My car has '+myCar.tyres+' tyres'+ ' and '+ myCar.windows+'
windows.')
console.log(myCar.getCarType())
console.log(myCar.electricCar)
//My car has 4 tyres and 6 windows.
//I own a very nice sedan.
//true
//My car has 4 tyres and 1 windows.
//I own a very nice convertible.
//true
```

#challenge:

Write yourself a virtual cat.

- Create a class that represents a cat. It should have properties for `tiredness`, `hunger`, `lonliness` and `happiness`
- Next, write methods that increase and decrease those properties. Call them something that actually represents what would increase or decrease these things, like "feed", "sleep", or "pet".

- Next, write a method that prints out the cat's status in each area. (e.g. Paws is really hungry, Paws is VERY happy.)
- Last, write a class of FlyingCat that extends the Cat class, adds a property for `flightDistance`, a method like “fly” that will affect the other properties tiredness and hunger.