JAVASCRIPT CLASSES

- The class keyword was introduced in ECMAScript 2015.
- However, they have not induced a new inheritance model the prototype inheritance model still holds

JAVASCRIPT CLASSES

```
class wahoo{
    constructor(leadership, grit){
        this.leadership = leadership
        this.grit = grit
    }
}
```

OVERRIDES THE CONSTRUCTOR IN OBJECT.PROTOTYPE

```
john = new wahoo(0.7,0.9)
console.log(john.grit)
```

USES THE NEW KEYWORD TO CALL THE CONSTRUCTOR

```
console.log(john.toString())
```

THE TOSTRING METHOD EXIST BECAUSE THE CLASS STILL INHERITS FROM OBJECT.PROTOTYPE

JAVASCRIPT CLASS EXPRESSION

- Class Expressions are another way to define classes

```
tribe = class{
    constructor(creativity, community){
        this.creativity = creativity
        this.community = community
    }
}
```

```
griffin = new tribe(0.9, 0.7)
console.log(griffin.community)
```

PRINTS OUT 0.7

```
console.log(griffin.name)
```

PRINTS OUT TRIBE

JAVASCRIPT PROTOTYPE METHODS

```
tribe = class{
    constructor(creativity, community){
        this.creativity = creativity
        this.community = community
    }
    //Getter
    get average(){
        return this.calAverage()
    }
    //Method
    calAverage(){
        return (this.creativity  + this.community)/2
    }

}
```

```
griffin = new tribe(0.9, 0.7)
console.log(griffin.average)
```

PRINTS OUT 8

GETTER CALLED WITH DOT NOTATION CAN BE THOUGHT OF AS A PROPERTY WITH CALCULATION

JAVASCRIPT STATIC METHODS

STATIC METHODS CAN BE CALLED WITH OUT INITIALIZING THE CLASS

```
class wahoo{
    constructor(leadership, grit){
        this.leadership = leadership
        this.grit = grit
    }

    static compareLeadership(wahoo1, wahoo2){
        return wahoo1.leadership === wahoo2.leadership
    }
}

john = new wahoo(0.7,0.9)
david = new wahoo(0.7, 0.3)
console.log(wahoo.compareLeadership(john, david))
```

PRINTS TRUE

CANNOT BE CALL ON A INSTANCE OF THE CLASS

JAVASCRIPT CLASSES AND EXTENDS

```
class Animal{
    constructor(name, age){
        this.name = name
        this.age = age
    }
    /** Animals age at different rates*/
    getOlder(years, factor){
        this.age += years * factor
    }
}

class Dogs extends Animal{
    getOlder(years){
        super.getOlder(years, 7)
    }
}

rex = new Dogs('rex', 0)
rex.getOlder(1)

console.log(rex.age)
```

CALLS THE METHOD IN THE SUPERCLASS

USES THE CONSTRUCTOR OF THE SUPERCLASS

PRINTS OUT SEVEN

JAVASCRIPT CLASSES AND EXTENDS

```
class Animal{
    constructor(name, age){
        this.name = name
        this.age = age
    }
    /** Animals age at different rates*/
    getOlder(years, factor){
        this.age += years* factor
    }
}

class Dogs extends Animal{
    constructor(name, age, species){
        super(name, age)
        this.species = species
    }
    getOlder(years){
        super.getOlder(years, 7)
    }
}

rex = new Dogs('rex', 0, 'german shepherd')
rex.getOlder(1)
```

IF THERE IS A CONSTRUCTOR IT MUST CALL SUPER BEFORE USING THE THIS KEYWORD

JAVASCRIPTS EXTENDS

- You can still extend classes without constructors. Because all classes inherit from Object.protytype. All classes have its default constructor

```
class Person{
    speak(){
        console.log("Hello World")
    }
}

class Father extends Person{
    constructor(name){
        super()
        this.name = name
    }
}

daniel = new Father('Daniel')
daniel.speak()
```

SUPER CALLS THE CONSTRUCTOR OF THE SUPER CLASS WHICH IS THE SAME AS OBJECT.PROTOTYPE

ANONYMOUS FUNCTIONS

NO NAME

```
career = function(last, increase){
    return last*(1+increase)
}
```

```
console.log(career(0.7, 0.1))
```

ANONYMOUS FUNCTIONS

FUNCTIONS ARE OBJECTS

FUNCTIONS CAN BE ASSIGNED AS PROPERTIES OF OBJECTS

```
wahoo = {
    skill: 0,
    grow: function(){
        console.log('growing')
    }
}
```

```
wahoo.grow
```

```
[Function: grow]
```

Gets the property

```
wahoo.grow()
```

Recall the getter don’t require bracket Notation

Bracket notation: Invokes the function

THE DEFAULT ARGUMENT PARAMETERS

MULTIPLE VARIABLE DECLARATIONS

```
sum = function(){
    let i , sum = 0
    for(i = 0; i < arguments.length; i+=1){
        sum += arguments[i]
    }
    return sum
}

console.log('The sum was ' + sum(1,2,3,4,5))
```

PRINTS 15

The arguments variable contains an array of all of the arguments passed to the function

DEFAULT ARGUMENTS

TWO ADDITIONAL DEFAULT ARGUMENTS ARE ALWAYS PASSED TO A FUNCTION: THIS AND ARGUMENTS

```
wahoo = {
    skill: 0,
    grow: function(){
        console.log('growing')
        skill = 12
        return(this.skill)
    }
}
```

Dependent on the innovation pattern

```
console.log(wahoo.grow())
```

PRINTS OUT: GROWING & 0

THE THIS PROPERTY REFERS TO THE OBJECT

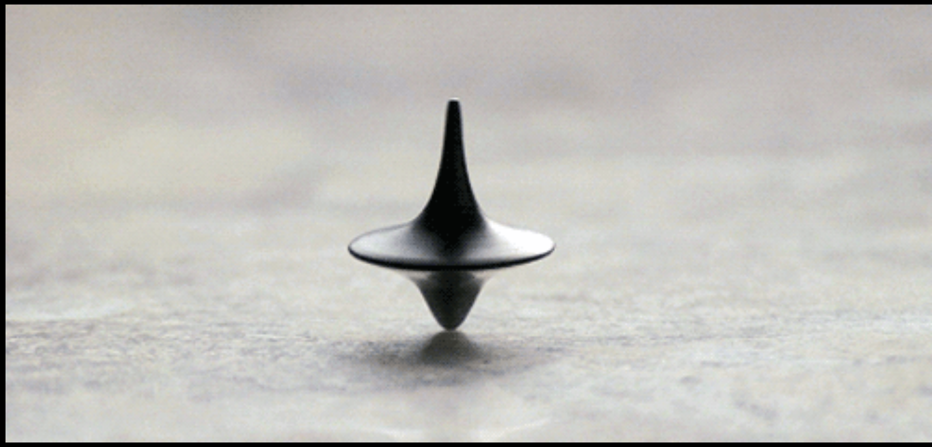
FUNCTION INVOCATION PATTERN

```
grow = function(){
    console.log('growing')
    skill = 12
    return(this.skill)
}
```

WITHIN THE SCOPE OF THE FUNCTION

```
console.log("result of grow" + " " + grow())
```

FUNCTION INCEPTION



A FUNCTION WITHIN A FUNCTION

```
function levelOne(){
    this.level = 1
    name = 'john'
     levelTwo = function(){
        this.level = 2
        console.log("In Level 2 " + this.level)
    }
    this.levelTwo()
    console.log("In Level 1 " + this.level)

}
levelOne()
```

RULE OF THUMB
THIS REFERS TO TOP
LEVEL INVOKING FUNCTION

PRINTS OUT IN LEVEL 2 2

PRINTS OUT IN LEVEL 1 2

ENCLOSING FUNCTIONS

- A Closure is a Javascript feature where an inner function has access to the outer functions variables
    - Inner function has access to it’s own scope
    - It has access to the outer function’s variables
    - It has access to the global variables

EXCEPTIONS

```
sum = function(){
    let i , sum = 0
    for(i = 0; i < arguments.length; i+=1){
        value = arguments[i]
        if( typeof value !== 'number' ){
            throw{
                name: 'TypeError',
                message: 'Type other than number found'
            }
        }
        sum += value
    }
    return sum
}

console.log('The sum was' + sum(1,2,3,4,'5'))
```

MODIFY THE SUM METHOD TO THROW SOMETHING WHEN A TYPE OTHER NUMBER IS PASSED IN

STOP AND THINK: FUNCTIONS AS ARGUMENTS

Remember functions are first class objects

```
tricky = {
    quad: function(double, x){
        return double(this.x) + double(x)
    },
    x: 3
}
```

WHAT GETS PRINTED OUT

```
result = tricky.quad(function double(x){
    x*= 2
    return x
},2)

console.log(result)
```

PRINTS 10