Functions Chaining

@Ammar Habib

swipe

What is functions chaining

Function chaining is a pattern in JavaScript where multiple functions are called on the same object consecutively. Using the same object reference, multiple functions can be invoked. It increases the readability of the code and means less redundancy.

```
person
    .sleep()  // is sleeping
    .wakeUp() // wake up
    .eat()    // is eating
    .study()  // is studying
```

person is an instance of an object that define methods

Why use functions chaining

In programming it is a commonplace to have actions that need to run in a defined series of steps. Creating a single function that define all these actions is usually a terrible idea so we write a number of functions that deal with individual actions

```
person
    .sleep()  // is sleeping
    .wakeUp() // wake up
    .eat()    // is eating
    .study()  // is studying
```

define actions and call methods in a series

Functions chaining can be creating by defining several methods in an object that return this which is an instance of the object so we can call another function from the object

Ways of creating functions chaining

1. object methods
2. class methods
3. functions prototype

1. Object Methods:

We define methods in an object which returns this (an instance of the object) as foloowing

```
const person = {
    sleep : function () {
         console.log('is sleeping')
         return this
    },
    wakeUp : function () {
        console.log('woked up')
        return this
    },
    eat : function () {
        console.log('is eating')
        return this
    },
    study : function () {
        console.log('is studying')
        return this
    }
}
```

of course methods can contain complicated logic to define each action

2. Class Methods:

We define methods in a class which returns this (an instance of the class) as foloowing

```
class Person {
    sleep(){
        console.log('is sleeping ');
        return this
    }
    wakeUp(){
        console.log('woked up');
        return this
    }
    eat(){
        console.log('is eating');
        return this
    }
    study(){
        console.log('is studying');
        return this
    }
}
```

3. Function Prototype

The prototype of a function is also called the signature of the function. The prototype data property of a Function instance is used when the function is used as a constructor with the new operator. It will become the new object's prototype

```
function Person() {}

Person.prototype.sleep = function(){
    console.log('is sleeping');
    return this
}
Person.prototype.wakeUp = function(){
    console.log('woke up ');
    return this
}
Person.prototype.eat = function(){
    console.log('is eating');
    return this
}
Person.prototype.study = function(){
    console.log('is studying');
    return this
}
```

```
const person = new Person()

person
    .sleep()      // is sleeping
    .wakeUp()     // woke up
    .eat()        // is eating
    .study()    // is studying
```