Javascript Basicis Notes Review

String properties and Method:

toUpperCase:

Change the String to upper case,

toLowerCase:

Change the String to lower case,

charAt(index):

Locate the character a the specific index of the String,

indexOf(string):

Locate the index of the specific String in a String,

trim():

To trim the wihtespace from the beginning,

starts(/ends)With(String):

Return if the String starts(or ends) with the specific String,

includes(String):

Return is the String contains the specific String,

slice(start,length):

Return the specific length of the String from the start index,

slice(start):

If you do not set a length ,it will slice all the rest string from this String,
and if you set the start number with a minus one,it will start to count from the end of this String, but there is one thing to note that from the end of the string, the start number is -1

Template Literals

This is a kind of expression in Javascript, with template literals you could use the specific objects or expressions directly without the concatnation of the string, and the method to use it is use `` to instead of using double quotation and single quotation, and use the use the ${}to include the expressions and objects.

Instance:

```
const name = "John";
const age = 22;
const sentence = `Hey it's ${name} and he is ${age} years old`;
const message = "Hey it's ${name} and he is ${age} years old";
console.log(sentence);
console.log(message);
function FullName({ firstName, lastName }) {
  const fullName = `${firstName} ${lastName}`;
  return fullName;
}
console.log(FullName({ lastName: "Yang", firstName: "Rongxin" }));
const add=`${2 + 2}`
console.log(add)
```

Then we will get the result like that:

```
Hey it's John and he is 22 years old
Hey it's ${name} and he is ${age} years old
Rongxin Yang
4
```

Array properties and methods

length:

This property will get the full length of the array

concat(array):

When we use A.concat(B), the content of the array B will be added to array Array

reverse():

Use this method we could reverse the whole order the this array

unshift(content):

Use this method we will add the element to the head of the array

shift():

Use this method we will remove the element from the head of the array

push(content):

Use this method we will add the element to the end of the array

pop():

Use this method we will remove the element from the end of the array

splce(start,length):

Use this method we could mutate the array, then we will get an array that contains the specific content from the old one

The deference between reference and value

When we assigining primitive data type value to a varible, any changes are made directly to that value, without affecting orginal value.But when we assigining non-primitive data type value to a varible is done by reference, so any changes will affect all the references

Here's the instance

```
//Change only the value
const number=1;
let number2=number;
number2=7;
console.log(`the first value is ${number}`);
console.log(`the second value is ${number2}`);
//Change the reference
let person={name:'bob};
let person2=person;
person2.name="susy";
console.log(`the first person is ${person.name}`);
console.log(`the second person is ${person2.name}`);
```

Here's the result

```
the first value is 1
the second value is 7
the first person is susy
the second person is susy
```

If we do want to just give person2 the value of the person, not the reference ,we could set person as one property of the person2, like this:

```
let person2={person}
```

The deferences between null and undifined

The undefined value means the javaScript could not find value for this,so it includes three conditions:

1. Variable without value
2. Missing function arguments
3. Missing object properties

Here's the example:

```
const people = {
  name: "tom",
};
sayHi = function (name) {
  return `Hello there! I am ${name}`;
};
let bye;
console.log(bye);
console.log(people.age);
console.log(sayHi());
```

The result:

```
undefined
undefined
Hello there! I am undefined
```

Besides the null is a value set by developer

Truthy and Falsy

In JavaScript, we could use True or False to represent this value is true or false, like this:

```
const true =true;
const false=false;
```

But in fact, we could use the String or a number to represent this too, like this :

```
const text="Hello";
if(text){
console.log(`Hey the value is Truthy`);
}
else{
console.log(`Hey the value is Falsy`);
}
```

The code above will result as the text is true.

And there are the conditions that is considered as falsy(The falsy conditions are less than truthy):

When the single quotation , double quotation and the back tick is empty, a number is equal to 0 or -0, a variable is NaN or undefined, and a string is false,

object.forEach(callback function)

This method is used to iterate the specific object elements(uesd to be an array), so it's invoked by an object. And it's argument is an function that describe how to iterate this object.

object.map(callback fuction)

The method forEach can only manipulate the data in itself, we could not return the value of in forEach, but the methods map could do that , to use map ,we will get an array when we return anything, and here's the example:

```
const people = [
  { name: "Tom", age: "22", sex: "Male" },
  { name: "Bob", age: "26", sex: "Male" },
  { name: "Jucy", age: "22", sex: "Female" },
];
let names = people.map(function (item) {
  return item.name;
});
console.log(names);
```

The code above will cause this result:

```
["Tom","Bob","Jucy"]
```

If you the method forEach, you will only get the undefined

object.filter(callback function)

Whth this method, we could return the objects that meet the condition we set, and here's the example:

```
const people = [
  { name: "Tom", age: "22", sex: "Male" },
  { name: "Bob", age: "26", sex: "Male" },
  { name: "Jucy", age: "22", sex: "Female" },
];
let person = people.filter(function (item) {
  return item.age < 25;
```

```
});
console.log(person);
```

```
[
    {
        "name": "Tom",
        "age": "22",
        "sex": "Male"
    },
    {
        "name": "Jucy",
        "age": "22",
        "sex": "Female"
    }
]
```

So we got this array that contains the objects meet our condition

onject.find(callback fuction)

This method has so many similarities with filter, but this method has a difference to it is it will only get the first onject, so this method is made to be found the unique object, like this:

```
const people = [
  { name: "Tom", age: "22", sex: "Male" },
  { name: "Bob", age: "26", sex: "Male" },
  { name: "Jucy", age: "22", sex: "Female" },
];
let person = people.find(function (item) {
  return item.sex === "Male";
});
console.log(person);
```

It will return the result:

```
{
    "name": "Tom",
    "age": "22",
    "sex": "Male"
}
```

obj.reduce(callback function(acc,curentItem){},initial value)

This method is more complicated than previous methods, there are two arguments in it, one for callback function and one for the value, and this value could be a number , an array or an object, whatever it is it could

effect the result. The first of all we focus on the fuction, in this function we found there are two parameters , the first one is the accumulation of the result, and the second one is the object we current iterate. And the accumulation is base on the second parameter in reduce method at the first time, here's the instance for the method:

```
const people = [
  { name: "Tom", age: "22", sex: "Male", sallery: 200 },
  { name: "Bob", age: "26", sex: "Male", sallery: 300 },
  { name: "Jucy", age: "28", sex: "Female", sallery: 500 },
  { name: "Kasha", age: "32", sex: "Female", sallery: 500 },
];
const dailySallery = people.reduce(function (total, cur) {
  console.log(`total: ${total}`);
  console.log(`current money:${cur.sallery}`);
  total += cur.sallery;
  return total;
}, 0);
```

Here's the result:

```
total: 0
current money:200
total: 200
current money:300
total: 500
current money:500
total: 1000
current money:500
```

Square Notation:

In javaScript we not only have the dot natation, but also the square which make us to build the dynamic properties for our object, for example:

```
const subject='math'
const favoriate={}
favoriate[subject]='some value'
console.log(favoriate)
```

We will get the result:

```
{
    "math": "some value"
}
```

If we change the subject to history, we will get the result:

```
{
    "history": "some value"
}
```