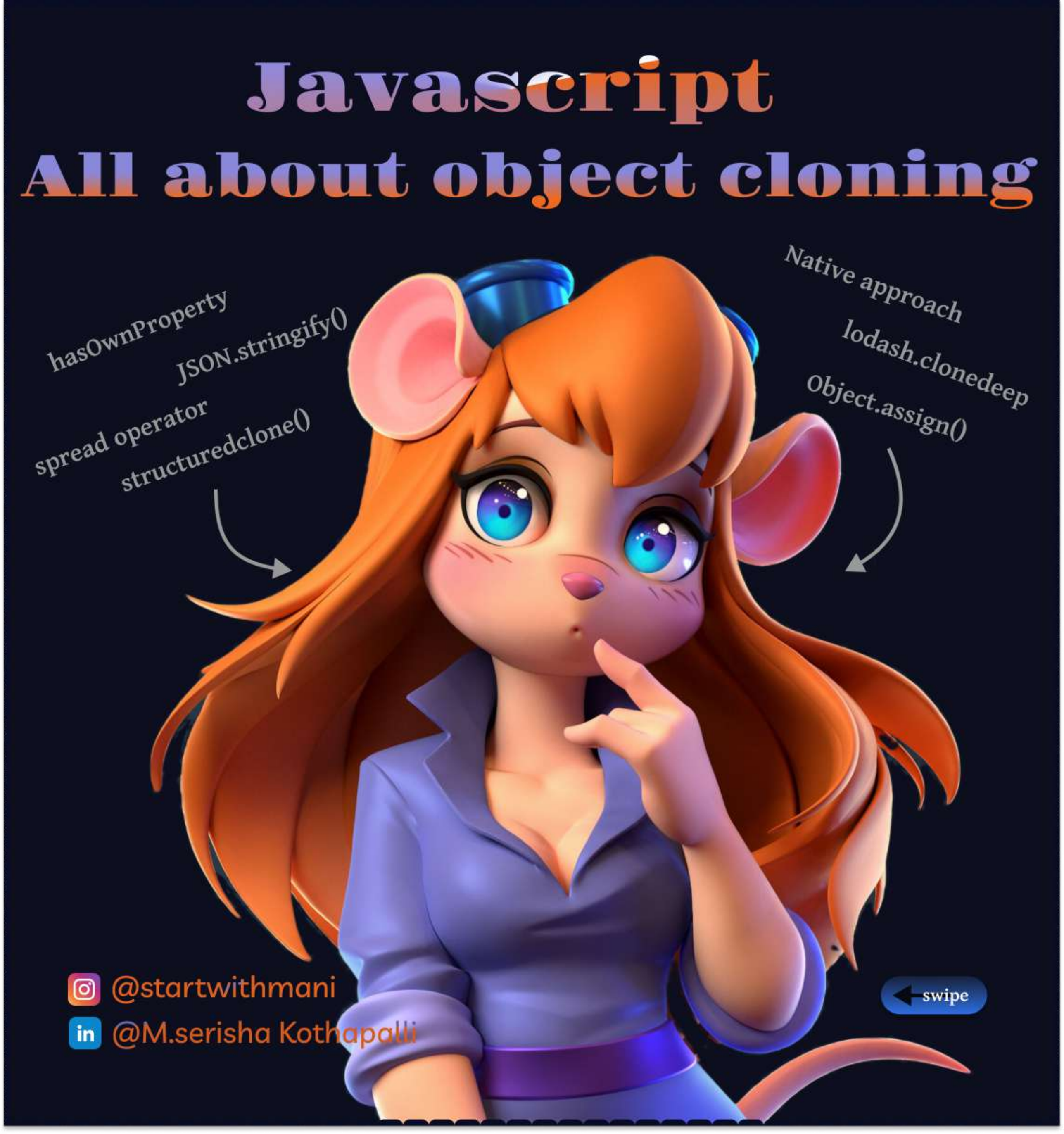
Javascript
All about object cloning



Cloning of objects:

JavaScript offers many ways of copying objects, but all of them do not provide a deep copy. Performing shallow copies is the default behavior in most of the cases. lets see some terminologies about objects. Make sure you are comfortable with all these before moving to next slide.....😊

```
let person = {
    name: 'Mani',
    address: {
        country: 'INDIA'
    },
    message: ()=> {
        console.log('Welcome Mani')
    }
};
```

direct properties /Top level properties

nested properties/ sub properties

The properties of a nested object

nested object??
An object (address) that is present inside another object (person).

method

An object property that is having function as value

@startwithmani

@M.serisha Kothapalli

swipe

Assignment operator

Copying of an object variable into other variable using assignment opeartor (=).In this method We have two variables, each making reference to the same object in memory.

```
let originalObject = {
    name: 'Mani',
    address: {
        country: 'INDIA'
    },
    message: ()=> {

      console.log('Welcome Mani')
    }
};

let clonedObject = person
clonedObject.name = 'Serisha'
console.log(originalObject.name)  //'Serisha'
```

Notice that property of original object is also changed on changing the cloned object property

This is called ***shallow copying*** as it is stil connected to original object. which we discussed in **previous posts**.....

@startwithmani
@M.serisha Kothapalli

swipe

Object.assign()

Object.assign() is used to copy the values and properties from one or more source objects to a target object.

Syntax:

const **copied** = **Object.assign(target, ...sources)**

target – target object to which values and properties are copied

sources – source object from which values and properties are copied

Pros:

- This method copies the direct properties and methods of an object

Cons:

- This method doesn't work for nested properties. Which means the changes made for nested property in cloned object changes the original object's nested property as well

@startwithmani
@M.serisha Kothapalli

swipe

```
let person = {
    name: 'Mani',
    address: {
        country: 'INDIA'
    }
};


let copiedPerson = Object.assign({}, person);

copiedPerson.name = 'Serisha';
copiedPerson.address.country = 'UK';

console.log(person.name);  //Mani
console.log(person.address.country);  //UK
```

disconnected

connected to original object

Notice in the above example the changes made for direct property (name) didn't affect the original object but for nested property there is a change in original object's nested property also .....so this also comes under ***shallow copying***

@startwithmani
@M.serisha Kothapalli

swipe

Using Spread operator:

Another way to copy objects in JavaScript is with the ES6 spread operator. Using the three dots (...) collects the values on the original object into another object. To have a very good understanding about this, lets categorise it into sections.

- Using spread operator for direct or top level /direct properties
- Using spread operator for nested properties .

For top level properties:

```
let originalObject = {
    name: 'Mani',
    instagram:'startwithmani'

};

let clonedObject = {...originalObject};
clonedObject.name = "serisha";

console.log(originalObject.name); //'Mani'
```

Notice there is no affect on original object for direct or top level properties.....

@startwithmani
@M.serisha Kothapalli

swipe

For nested properties:

Many people say that spread operator doesn't work for nested objects...But that is not the case. To make a complete deep copy with the spread operator, we'll have to write some additional code.

Now lets consider an object with nested properties and know how it works. For any **nested object**, we must also **spread** that **sub-object** before making changes to any of its sub properties.

If you **don't spread** the **nested object** that endsup performing shallow copying....means, the changes made for nested object property with out spreading will also change the original one

Let's have better understanding with an example.....

@startwithmani
@M.serisha Kothapalli

swipe

For nested properties:

```
let originalObject = {
    name: 'Mani',
    instagram:'startwithmani',
    address : {
        country : 'INDIA'
    }

};

let clonedObject = {...originalObject,
                    address : {...originalObject.address}
                   };
clonedObject.address.country = "UK";

console.log(originalObject.address.country);  // 'INDIA'
```

Here the address is an **nested object**. so to perform **deep copy** for nested object, you need **spread** that object as well....Doing that as you can notice from above example country property in **original object** stays same and is **unaffected**

@startwithmani
@M.serisha Kothapalli

swipe

JSON.stringify() & JSON.parse()

- The JSON.stringify() method takes in an object and creates a JSON string from it.
- The JSON.parse() method parses a string and returns a JavaScript object.

We can combine these two to create an copy of an object....Lets check that as well.....😁

```
let originalObject = {
    name: 'Mani',
    address : {
        country : 'INDIA'
    }

};

let clonedObject = JSON.parse(JSON.stringify(originalObject));

clonedObject.name = "Serisha";
clonedObject.address.country = "UK";

console.log(originalObject.name); //'Mani'
console.log(originalObject.address.country);  // 'INDIA'
```

Notice this it works both for direct and nested properties

@startwithmani
@M.serisha Kothapalli

swipe

Yay !.....We finally found something which works both for nested and direct properties ....Great! . But do you know still it doesn't work in all cases,which is a bit disappointing.....😕

Let us consider an example with method in an object...and try to copy that object.

Notice this, your method is not even copied into your clonedObject.

```
let originalObject = {
    name: 'Mani',
    message:() => console.log(`${this.name} is from
${this.address.country}`)

};

let clonedObject = JSON.parse(JSON.stringify(originalObject));


console.log(clonedObject);  //{ "name": "Mani")
```

To summarize although the JSON.stringify works on properties. It doesn't work for methods, it doesn't copy your functions.....

@startwithmani
@M.serisha Kothapalli

swipe

structuredClone():

The method is also used for object cloning. It works for direct properties and nested properties but this method doesn't work for methods in objects. It throws a DOMException if we try to do so for methods as shown below.....

```
let originalObject = {
    name: 'Mani',
    address : {
        country:'india'
    },
    message: () => console.log(`${this.name} is from
${this.address.country}`);

};

let clonedObject = structuredClone(originalObject);

console.log(originalObject);
```

Since we are trying to use for method message...Notice the output ...it throws DOM exception.

```
▸Uncaught DOMException: Failed to execute 'structuredClone' on 'Window': ()=>
console.log("hello world") could not be cloned.
    at snippet:///Script%20snippet%20%237:9:20
```

@startwithmani
@M.serisha Kothapalli

swipe

Using hasOwnProperty() by iterating:

This method is used by iterating through a source object's properties and copying them one after the other to a target object. But unfortuantely this also doesn't work for nested properties....

```
let originalObject = {
    name: 'Mani',
    address : {
        country:'india'
    }
};

let clonedObject =  {};
for (let key in originalObject) {
  if (originalObject.hasOwnProperty(key)) {
  clonedObject[key] = originalObject[key];
 }
}

clonedObject.address.country = 'UK';

console.log(originalObject.address.country);  // 'UK'
```

Notice here this changed the nested property of original object as well

@startwithmani
@M.serisha Kothapalli

swipe

Summary:

I know this is much of it.....Huh....There are stil lot more ways for cloning.

One of the preffered way to perform deep copy is by utilizing external dependency **lodash** libarary. Lodash provides a utility method **_.cloneDeep()** for deep cloning of objects in JavaScript.

I don't wanted get into that at this point and make you confused..I will discuss clearly in upcoming posts. So Just to conclude there are plenty of ways to clone an object with it's own Pros and cons. You need to pick the one which would suffice your requirement.....

Stay tuned.....😁

@startwithmani
@M.serisha Kothapalli

swipe

Did you find it helpful??

Like this post!

Share with your friends

Save it for later

Follow for more!

@startwithmani

@M.serisha Kothapalli