JavaScript Activites

30/10/2019

* 2.5 hours, react tutorial by Mosh Hamedani
  + bind()
* 50 minutes, JavaScript for react developers by Mosh Hamedani
* 48 minutes, JavaScript for beginners by Mosh Hamedani
* 62 minutes, Object-oriented Programming in JavaScript by [Mosh Hamedani](https://www.youtube.com/watch?v=PFmuCDHHpwk)
  + JavaScript Abstraction by Raghav Pal [10/10]
  + call(), apply(), bind() by [Tech Sith](https://www.youtube.com/watch?v=c0mLRpw-9rI) [bind wasn’t well explained]
* 3.5 hours, Learn JavaScript full course for beginners by [freeCodeCamp.org](https://www.youtube.com/watch?v=PkZNo7MFNFg)
  + for … in & for … of by [freeCodeCamp](https://www.youtube.com/watch?v=a3KHBqH7njs)
  + [JavaScript rest Operators](https://www.youtube.com/watch?v=j3bXlt3NOZA)
  + JavaScript getters and setters by Mosh Hamedani
  + Import & export by Beau
  + JavaScript Constructor Function by Net Ninja
  + The rest parameter syntax by ddcode
* JavaScript DOM crash course – Part 1 by Traversy Media
* JavaScript DOM crash course – Part 2 by Traversy Media
* JavaScript DOM crash course – Part 3 by Traversy Media
* JavaScript DOM crash course – Part 4 by Traversy Media
* JavaScript higher order functions by Traversy Media

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Abstraction in JS

Find the clear and concise explanation in this video:

<https://www.youtube.com/watch?v=jM0WcyQWMSM>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Value vs Reference Types

<https://www.youtube.com/watch?v=PFmuCDHHpwk>

31:09: Value vs Reference Types

Primitives types are copied by their value.

Reference types are copied by their reference.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Accessing object properties with variables

let obj = {

  12: "Twelve Grade",

  13: "University Grade",

  14: "Bachelor",

  15: "Master"

};

let res = obj[12];

console.log(res);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# reduce()

<https://www.youtube.com/watch?v=g1C40tDP0Bk>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Rest operators

<https://www.youtube.com/watch?v=PkZNo7MFNFg&t=107s>

([2:54:00](https://www.youtube.com/watch?v=PkZNo7MFNFg&t=10440s)) 118. Use the Rest Operator with Function Parameters

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Object Literals, Constructor Function and Factory Function

Find the clear and concise explanation in this video:

<https://www.youtube.com/watch?v=PFmuCDHHpwk>

* 11:53: Object Literals
* 14:58: Factories
* 17:50: Constructors

If we use ‘this’ with ‘new’ operator, we refer to that as constructor function.

If we ‘return’ something, we refer to that as factory function.

// factory function

function circle2(radius) {

  // in factory function we return

  return {

    radius: radius, // we used ':' not '='

    draw: function() {

      console.log("draw");

    }

  };

}

const circle2\_instance = circle2(20);

console.log(circle2\_instance.radius); // 20

// constructor function

function Circle3(radius) {

  this.radius = radius;

  this.draw = function() {

    console.log("draw");

  };

}

const circle3\_instance = new Circle3(30);

console.log(circle3\_instance.radius); // 30

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Constructor Functions

They are used to make objects.

[*https://www.youtube.com/watch?v=23AOrSN-wmI&t=7s*](https://www.youtube.com/watch?v=23AOrSN-wmI&t=7s)

function Car(maxSpeed, driverName) {

  this.maxSpeed = maxSpeed;

  this.driverName = driverName;

  this.drive = function() {

    console.log(`${this.driverName} is driving!`);

  }; // put a ';' not ','

  this.carKMs = function(distance) {

    // see below, how how we passed args

    console.log(`His car is ${distance} KMs.`);

  };

}

const car1 = new Car(240, "John Eric");

console.log(car1.maxSpeed);

console.log(car1.driverName);

car1.carKMs(2000);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# hasOwnProperty()

hasOwnProperty returns a boolean value indicating whether the object on which you are calling it has a property with the name of the argument. For example:

var x = {

  y: 10

};

console.log(x.hasOwnProperty("y")); //true

console.log(x.hasOwnProperty("z")); //false

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Factory Functions

[*https://www.youtube.com/watch?v=ImwrezYhw4w*](https://www.youtube.com/watch?v=ImwrezYhw4w)

// Class vs Factory Functions

class Dog {

  constructor() {

    this.sound = "Wooof";

  }

  bark() {

    console.log(this.sound);

  }

}

const husky = new Dog();

husky.bark();

// $("button.btn").click(husky.bark.bind(Dog)); // we've problem using this

// $("button.btn").click( \_ => husky.bark() );

// Above class in Factory Function

const dog = () => {

  const sound = "Woof woof ... ";

  return {

    talk: () => {

      console.log(sound);

    }

  };

};

const sniffy = dog();

sniffy.talk();

// $('button.btn').click(sniffy.talk); // we don't have problem using 'this'

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Object.freeze()

<https://www.youtube.com/watch?v=PkZNo7MFNFg&t=107s>

[2:44:52](https://www.youtube.com/watch?v=PkZNo7MFNFg&t=9892s) 113. Prevent Object Mutation

// A frozen object can no longer be changed; freezing an object

// prevents new properties from being added to it, existing

// properties from being removed, prevents changing the enumerability,

// configurability, or writability of existing properties, and prevents

// the values of existing properties from being changed.

const PI = {

  val: 3.14

};

Object.freeze(PI);

function func() {

  PI.val = 33;

  return PI.val;

}

console.log(PI.val);

const res = func();

console.log(res);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# import / export ES6

<https://www.youtube.com/watch?v=Jqn_wjkSZwo>

# defineProperty(), Setters and Getters in OOP

<https://www.youtube.com/watch?v=PFmuCDHHpwk>

51:55: Getters and Setters

function Employee(name, position) {

  this.name = name;

  let monthlyBonus = 300;

  let salary = 2000;

  this.position = position;

  this.work = function() {

    console.log(`Working as ${this.position}!`);

  };

  Object.defineProperty(this, "monthlyBonus", {

    get: function() {

      return monthlyBonus;

    },

    set: function(value) {

      monthlyBonus = value;

    }

  });

  Object.defineProperty(this, "salary", {

    get: function() {

      return salary;

    },

    set: function(val) {

      salary = val;

    }

  });

}

emp1 = new Employee("John Eric", "CEO");

console.log("Name: ", emp1.name);

emp1.salary = 5000;

console.log("Salary: ", emp1.salary); // 5000

emp1.monthlyBonus = 1000;

console.log("bonus: ", emp1.monthlyBonus); // 1000

console.log(emp1.work()); // Working as CEO

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

let person = {

    name:'John Doe',

    age:25,

    get get\_info(){

        return `${person.name} is ${person.age} years old`;

    },

    set get\_info(val){

        person.name, person.age = val.split(' ')

    }

}

person.name = 'Smith';

person.age = 55;

console.log(person.get\_info);

# Fetching data from an API

<https://scrimba.com/p/p7P5Hd/c79Jask> [Best One]

class Main extends React.Component {

  constructor(){

    super();

    this.state = {

      loading: false,

      person : {}

    }

  }

  componentDidMount(){

      this.setState({loading:true});

      fetch("https://swapi.co/api/people/1/")

        .then(response => response.json())

        .then(data => {

            this.setState({loading:false});

            this.setState({ person: data })

        });

  }

  render() {

    return (

      <h1> {this.state.loading ? "loading ... " :  this.state.person.name }</h1>

    );

  }

}

ReactDOM.render(<Main />, document.getElementById('root'));