Date: 4-27-2020

Default parameters in Functions:
Definition: while defining the fucnts, few parameters are initialized to default values, That concept is called default parameters.

Introduced in ES6 version

Example 1:

```
function fun_one(arg1 = "ReactJs", arg2 = "AngularJs", arg3 = "MongoDB") {
    console.log(arg1, arg2, arg3);
}
fun_one(); //ReactJs AngularJs MongoDB
fun_one("Java"); //Java AngularJs MongoDB --
                   only arg1 will be changed. other remain default values
fun_one(undefined, undefined, undefined); //ReactJs AngularJs MongoDB, undef
                                          cannot change the default values
fun_one(null, null, null); //null null null, null will override default values
```

Example 2: (Combination of Regular parameters, Default parameters)

```
function fun_one(arg1, arg2 = "AngularJs", arg3) {
    console.log(arg1, arg2, arg3);
}
fun_one(); //undefined AngularJs undefined --- "Variable Hoisting"
fun_one("Java"); // Java AngularJs undefined --
                 only ar2 will be changed. other remain default values
fun_one(undefined, undefined, undefined); //undefined AngularJs undefined, undef
                 cannot change the default values
fun_one(null, null, null); //null null null, null will override default values
```

What is Variable hoisting?

If we are calling regular parameter without defining value, then undefined will be assigned to that variable. This is called Variable hoisting.

Example 3: (combination of Regular parameters, Default Parameters , Rest Paramets)

```
function fun_one(arg1, arg2 = "AngularJs", ...arg3) {
    console.log(arg1, arg2, arg3);
}
fun_one(); //undefined AngularJs [ ]
fun_one("Java", undefined, "Html", "css"); // Java AngularJs [ 'Html', 'css' ] --
                            only arg2 will be changed. other remain default values
fun_one(undefined, undefined, undefined); //undefined undefined [undefined],
                                    undef cannot change the default values
fun_one(null, null, null); //null null null, null will override default values
```

Arrow Funcitons:

Introduced in ES6, represent using => symbol

Also called as Fat Arrow function / Anonymous function (function without name is called as anonymous function)

When to use Arrow Functions:

- Handle events (click, touch , mouse over etc…) and bind them (events)
- Application performance is better compared to regular function (Theoretically)
- Object creations for functions is NOT possible in Arrow Functions i.e Representation of Data is NOT possible
- If we want to read response from server – Arrow functions are suggestible functions
  Syntax:
  Let variableName = (arguments …) => {…Business Logic….}

Example

```
let var1 = () => {
    console.log("Hello World");
}
console.log(var1); //[Function: var1]
console.log(var1()); // Hello World
```

Date: 28-04-2020

Arrow functions recap

Example in usage of arrow functions in html (filename.html):

```
<!DOCTYPE html>
<html>
<head>
    <meta charset='utf-8'>
    <meta http-equiv='X-UA-Compatible' content='IE=edge'>
    <title>Arrow Functions</title>
    <script>
        let my_fun = () => {
            return "Hello World....!"
        }
        console.log(my_fun); // will give function definition --  NOT Value
        console.log(my_fun()); // will give its value ( Business Logic )
    </script>
</head>
<body>
</body>
</html>
```

o/p:

fileName.html:16 () => {

return "Hello World....!"

}

fileName.html:17 Hello World....!

<u>Nested Arrow Functions:</u>

```
    <script>
        let my_fun = () => {
            return () => {
```

```
                return "Hello World....!"
            }
        }
        console.log(my_fun); // outer function definition
        console.log(my_fun()); // inner function definition
        console.log(my_fun()()); // inner function result…!
    </script>
```

O/p:

```
() => {
        return () => {
            return "Hello World....!"
        }
    }
defaultPar.html:16 () => {
            return "Hello World....!"
        }
defaultPar.html:17 Hello World....!
```

<u>Arrow Functions with arguments:</u>

```
<script>
        let my_fun = (arg1, arg2) => {
            console.log(arg1, arg2);
        }
        my_fun("ReactJs", "AngularJs");
    </script>
```

o/p:

ReactJs AngularJs

Arrow functions as arguments

```
<script>
    let asyncFun = (successResponse, errorResponse) => {
        console.log(successResponse); // Returns function definition
        console.log(successResponse()); // Return function output

        console.log(errorResponse); // Returns function definition
        console.log(errorResponse()); // Returns function output
    }
    asyncFun(
            () => {
                return "Success Message";
            }, // Argument 1

            () => {
                return "Error Message";
            } // Argument 2

        ) // Function ends
</script>
```

o/p:

```
() => {

        return "Success Message";

    }

fileName.html:11 Success Message

fileName.html:13 () => {

        return "Error Message";

    }

fileName.html:14 Error Message
```

Push arrow function definition to array upto 5 elements

```
<script>
        function myFun() {
            let myarray = [];
            for (let i = 0; i < 5; i++) {
                myarray.push(
                    () => {
                        return "Welcome " + i + "th time";
                    }
                )
            }
            for (let i = 0; i < 5; i++) {
                console.log(myarray[i]());
            }
        }
        myFun();
    </script>
```

o/p:

defaultPar.html:19 Welcome 0th time

defaultPar.html:19 Welcome 1th time

defaultPar.html:19 Welcome 2th time

defaultPar.html:19 Welcome 3th time

defaultPar.html:19 Welcome 4th time

<u>setTimeOut()</u>

Execute a function after a particular time…….

Syntax : setTimeout(functionName(), time_in_mSec's)

Example:

```
    <script>

        myFun = () => {
            console.log("Hello..");
```

```
        }
        setTimeout(myFun, 5000);
    </script>
```

Example 2: arrow function definition as parameter for setTimeout

```
<script>
        setTimeout(() => {
            console.log("Hello..");
        }, 5000);
    </script>
```

Example 3: Combination of regular statements with setTimeouts

```
    <script>
        console.log("Hello 1");
        setTimeout(() => {console.log("Hello 2");}, 5000);
        console.log("Hello 3");
    </script>
```

o/p:

Hello 1

Hello 3

(After 5 seconds) Hello 2

//This mechanism is called **"Event loop mechanism".** Hello2 will be executed in parallel. This is secondary thread.

Example 3: Combination of regular statements with setTimeouts with 0 priority

```
<script>
    console.log("Hello 1");
    setTimeout(() => {console.log("Hello 2");}, 0);
    console.log("Hello 3");
</script>
```

<u>o/p:</u>

Hello 1

Hello 3

Hello 2 executed by secondary thread.

Date: 29-04-2020

<u>setInterval()</u>

Execute a function for every interval of time.

Example

```
let myFun = () => {
    console.log(" Hello world..");
}
setInterval(myFun, 1000);
```

o/p:

prints Hello world for every one second.

Example 2:

```
let myFun = () => {
    console.log(new Date().toLocaleTimeString());
}
```

```
setInterval(myFun, 1000);
```

o/p: Displays current time for every 1 second.

Assignment: execute setInterval function only 5 times…

Solution:

```
let count = 0;
let timerId = 0;
let myFun = () => {
    console.log(new Date().toLocaleTimeString());
    count++;
    if (count > 5) {
        clearInterval(timerId);
    }
}
timerId = setInterval(myFun, 1000, 3);
```

o/p:

prints time only 5 times.

<u>Constructor Funtions:</u>

Used to create objects, using “new” keyword

“this” keyword is used to refer current class content

Object will be created in heap memory

Default Constructor Example:

```
function myConstructorFunction() {
    this.sub_one = "ReactJs";
    this.sub_two = "NodeJs";
    this.sub_three = "AngularJs"
```

```
}

let myObj = new myConstructorFunction();
console.log(myObj.sub_one, myObj.sub_two, myObj.sub_three);
```

o/p: ReactJs NodeJs AngularJs

Parameterized Constructor Example:

```
function myConstructorFunction(sub_one, sub_two, sub_three) {
    this.sub_one = sub_one;
    this.sub_two = sub_two;
    this.sub_three = sub_three;
}

let myObj1 = new myConstructorFunction("ReactJs", "AngularJs", "NodeJs");
let myObj2 = new myConstructorFunction("Java", "Adv.Java", "Spring");
console.log(myObj1.sub_one, myObj1.sub_two, myObj1.sub_three);
console.log(myObj2.sub_one, myObj2.sub_two, myObj2.sub_three);
```

o/p:

ReactJs AngularJs NodeJs

Java Adv.Java Spring

<u>Define Functions inside Constructor Functions:</u>

```
function myConstructorFunction() {
    this.MEAN = "MEAN Stack";
    this.MERN = "MERN Stack";
    this.MAVN = "MAVN Stack";

    this.meanFunction = function() {
```

```
        return this.MEAN;
    };

    this.mernFunction = function() {
        return this.MERN;
    };

    this.mavnFunction = function() {
        return this.MAVN;
    };
}

let myObj = new myConstructorFunction();
console.log(myObj.meanFunction(), myObj.mernFunction(), myObj.mavnFunction());
```

o/p: MEAN Stack MERN Stack MAVN Stack

<u>Function passed to another function as object / argument:</u>

```
function funOne(arg1) {
    this.test = arg1;
};

function funtwo() {
    this.test = "Welcome to constructor function..";
};

let obj1 = new funOne(new funtwo());
console.log(obj1.test.test);
```

o/p:

Welcome to constructor function.

Nested Functions:

```
function function1() {
    this.function2 = () => {
```

```
        console.log("Hello world..");
    }
}
let myVar1 = new function1();
myVar1.function2();
```

o/p:

Hello world..

Date: 30-04-2020

Constructor functions recap

<u>Adding properties(Variables) to constructor functions dynamically</u>

- *<u>Prototype</u>* is predefined keyword – used to add properties and functions to constructor function

Example:

```
function funcOne() {

}
funcOne.prototype.wish = "Hello world.."
let myObj = new funcOne();
console.log(myObj.wish);
```

o/p: Hello World

<u>Adding functions to constructor functions dynamically</u>

```
function myFunction() {

}
myFunction.prototype.fun_one = function() {
    return " From Function One";
};
myFunction.prototype.fun_two = function() {
    return " From Function Two";
```

```
};
myFunction.prototype.fun_three = function() {
    return " From Function Three";
};
let myObj = new myFunction();
console.log(myObj.fun_one());
console.log(myObj.fun_two());
console.log(myObj.fun_three());
```

o/p:

From Function One

From Function Two

From Function Three

<u>Combination of properties and functions to constructor functions dynamically</u>

```
function myFunction() {

}
myFunction.prototype.es6 = "ES6";

myFunction.prototype.fun_one = function() {
    return ` From Function One  ${this.es6 }`;
};

let myObj = new myFunction();
console.log(myObj.fun_one());
```

o/p:

From Function One ES6

→Inheritance is called as prototype chaining in JS

→Getting properties and functions from parent class is called as prototype chaining.

→Child and parent class relationship can be made using Object.create(ParentClass)

Example:

```
function mean() {

}
mean.prototype.mean = "MEAN Stack";

function mern() {

}
mern.prototype = Object.create(mean.prototype); //Mean is Parent class, Mern is c
hild class
mern.prototype.mern = "MERN Stack"

function mevn() {

}
mevn.prototype = Object.create(mern.prototype); //Mern is Parent class, Mevn is c
hild class
mevn.prototype.mevn = "MEVN Stack";

let mevnObj = new mevn();
console.log(mevnObj.mean, mevnObj.mern, mevnObj.mevn);
```

o/p: MEAN Stack MERN Stack MEVN Stack

mean (parent)

↓

mern (child to mean, parent to mevn)

↓

mevn (child to mern and mean)

This process is called prototype chaining in JS.

Date: 1-05-2020

Console.dir(ObjectName) : is used to see the internal structure of object.

Example:

Note: paste below code in a html file and then test.

```
<script>
    function myFunction() {
        this.variable1 = "variable 1";
        this.variable2 = "variable 2";
        this.variable3 = "variable 3";
    }
    myFunction.prototype.variable4 = "variable 4"
    myFunction.prototype.variable5 = "variable 5"
    console.dir(myFunction);
</script>
```

o/p:

```
ƒ myFunction()

    1.  arguments: null
    2.  caller: null
    3.  length: 0
    4.  name: "myFunction"
    5.  prototype:
            1.  variable4: "variable 4"
            2.  variable5: "variable 5"
            3.  constructor: ƒ myFunction()
            4.  __proto__: Object
    6.  __proto__: ƒ ()
    7.  [[FunctionLocation]]: sampleHTML.html:12
    8.  [[Scopes]]: Scopes[1]
```

Note: Arrow Functions cannot have Prototype.

- ➔ Object.prototype is the parent for all custom functions.
- ➔ All the functions available in Object.prototype are available in custom Fun's

Eg: bind(), toString(), apply() etc etc etc…

For predefined functions:

- ➔ Date.prototype is Parent for Date
- ➔ Time.prototype is parent for Time

Date: 02-05-2020

<u>Optional Parameters in Funcitons</u>

While calling functions, few parameters are optional. representation "?" Symbol

Introduced in ES6

Will work in TypeScript Environment

File extension is .ts

<u>Installing Typescript:</u>

npm install –g typescript@latest

npm : Node Packaging Manager

- ➔ Npm present in NodeJs

-g : Global installation

Type script is programming Language.

"tsc" compiler is used to compile TypeScript programs.

Equivalent JS file gets generated (in current directory) after successful compilation of .ts file: this process is called as "Transpilation".

demo.ts $\xrightarrow{\text{Transpiled}}$ demo.js

Example:

```
function myFunction(arg1?:String, arg2?:string,arg3?:string){
    console.log(arg1,arg2,arg3);
}

myFunction(); //undefined undefined undefined - Variable Hoisting
myFunction("ReactJs"); //ReactJs undefined undefined
myFunction("ReactJs","AngularJs","Typescript"); //ReactJs AngularJs Typescript
myFunction("ReactJs",undefined,"AngularJs"); //ReactJs undefined AngularJs
myFunction(undefined,undefined,undefined); //undefined undefined undefined
myFunction(null,null,null); //null null null
```

Combination of Regular parameter and optional parameter

Note: Optional parameter should be in last position in this combination.

Example:

```
function myFunction(arg1:string, arg2?:string,arg3?:string){
    console.log(arg1,arg2,arg3);
}

// myFunction(); //Compilation Error - Expected 1 or 2 parameters but got zero
myFunction("ReactJs"); //ReactJs undefined undefined
myFunction("ReactJs","AngularJs","Typescript"); //ReactJs AngularJs Typescript
myFunction("ReactJs",undefined,"AngularJs"); //ReactJs undefined AngularJs
myFunction(undefined,undefined,undefined); //undefined undefined undefined
myFunction(null,null,null); //null null null
```

<u>Combination of Regular Parameter + Optional Parameter + Default Parameter</u>

```
function myFunction(arg1:string, arg2?:string,arg3:string="Hello3"){
    console.log(arg1,arg2,arg3);
}

//myFunction(); //Compilation Error - Expected 1 or 2 parameters but got zero
myFunction("ReactJs"); //ReactJs undefined Hello3
myFunction("ReactJs","AngularJs","Typescript"); //ReactJs AngularJs Typescript
myFunction("ReactJs",undefined,"AngularJs"); //ReactJs undefined AngularJs
myFunction(undefined,undefined,undefined); //undefined undefined Hello3
myFunction(null,null,null); //null null null
```

3-5-2020 - Sunday

Date: 4-5-2020

JSON

- ➔ **J**ava**S**cript**O**bject**N**otation also called as **J**ava**S**cript**O**bjects
- ➔ Light weight
- ➔ Parsing(Reading) of json is easy compared to XML

  Representation

  Objects – { }

  Arrays – [ ]

  Data – key and Value pairs

  Key and value – separated by colon (:)

  Key and value pairs – separated by comma (,)

Example:

```
let obj = {
    sub_one : "Nodejs",
    sub_two : "ReactJs",
    sub_three : "AngularJs"
}
console.log(obj.sub_one); //NodeJs
console.log(obj.sub_two); //ReactJs
console.log(obj.sub_three);//AngularJs
```

Points to Remember:

- ➔ Key should NOT start with Number, Even inside “ ” eg: “1” is NOT valid
- ➔ Key should NOT be reserved JS keywords (like var, let etc)
- ➔ If keys are Duplicated – Old keys will be replaced by new key
- ➔ Key can be “[ ]” or “{ }” (inside quotes). It is valid. But cannot be accessible directly, it can be accessible using loops.
- ➔ Key cannot be [ ] or { } – without quotes – it is invalid
- ➔ Key can be null or undefined – it is valid

Using loops to iterate json :

```
let obj = {
    sub_one : "Nodejs",
    sub_two : "ReactJs",
    sub_three : "AngularJs"
}
```

```
for(let key in obj){
    console.log(key,obj[key]);

}
```

o/p:

sub_one Nodejs

sub_two ReactJs

sub_three AngularJs

➔ Data type of a variable can be identified using “type of” operator.

Nested Json:

```
let obj1 = {
    obj2: {
        obj3: {
            sub_one: "Nodejs",
            sub_two: "ReactJs",
            sub_three: "AngularJs"
        }
    }
}

for (let key in obj1.obj2.obj3) {
    console.log(key, obj1.obj2.obj3[key]);

}
```

o/p:

sub_one Nodejs

sub_two ReactJs

sub_three AngularJs

Example:

Functions inside json.

```
let obj1 = {
    obj3: login
}
function login(){
console.log("Hello world..");

}

obj1.obj3(); //Function output : Hello world..
obj1.obj3; // Function definition
```

Example:

```
let obj1 = {
    oracle: oracle,
    mongoDb:mongoDb,
    mysql:mySql
}
function oracle() {
    return "oracle connection soon..";
}

function mongoDb() {
    return "mongoDB connection soon..";
}
function mySql() {
    return "mysql connection soon..";
}
console.log(obj1.oracle); //[Function: oracle]
console.log(obj1.oracle()); //oracle connection soon..

console.log(obj1.mongoDb); //[Function: mongoDb]
console.log(obj1.mongoDb()); //mongoDB connection soon..

console.log(obj1.mysql); //[Function: mySql]
console.log(obj1.mysql()); //mysql connection soon..
```

Date: 5-5-2020

Delete a particular key from json

```
let myJson = {
    obj1:"Object one",
    obj2:"Object two",
    obj3:"Object three",
    obj4:"Object four"
}


console.log(myJson);
//{obj1:'Object one',obj2:'Object two',obj3:'Object three',obj4:'Object four}

delete myJson.obj4;

console.log(myJson);
//{ obj1: 'Object one', obj2: 'Object two', obj3: 'Object three' }
```

Freeze():

Values cannot be changed.

Object.freeze(…)

Example:

```
let myJson = {
    obj1:"Object one",
    obj2:"Object two",
    obj3:"Object three",
    obj4:"Object four"
}

console.log(myJson);
//{obj1:'Object one', obj2:'Object two', obj3:'Object three', obj4:'Object four'}
Object.freeze(myJson);
myJson.obj1="New Object one";
//{obj1:'Object one', obj2:'Object two', obj3:'Object three', obj4:'Object four'}
```

Seal()

Data can be modified, but new data cannot be added.

Example:

```
let myJson = {
    obj1:"Object one",
    obj2:"Object two",
    obj3:"Object three",
    obj4:"Object four"
}

console.log(myJson); //{obj1: 'Object one',    obj2: 'Object two',    obj3: 'Obje
ct three',    obj4: 'Object four'  }
Object.seal(myJson);
myJson.obj1="New Object one";
console.log(myJson);
//{ obj1: ' New Object one ', obj2: 'Object two', obj3: 'Object three' }
myJson.obj5:"Object five" //Compilation error
```

DefineProperty()

→Add new property to object

Example:

```
let myJson={
    obj1:"object one",
    obj2:"object two",
    obj3:"object three"
}
Object.defineProperty(myJson,"obj4",{value:"Object four",writable:true,enumerable
:true});
console.log(myJson);
//{obj1: 'object one', obj2: 'object two', obj3: 'object three',obj4: 'Object fou
r'}
```

Note: If writable is set to false, that variable cannot be modified.

DefineProperties()

Example:

```
let myJson = {
    obj1: "object one"
```

```
}
Object.defineProperties(myJson, {
    "obj2": { value: "Object two", writable: true, enumerable: true },
    "obj3": { value: "Object three", writable: true, enumerable: true },
});
console.log(myJson);

O/P:
{ obj1: 'object one', obj2: 'Object two', obj3: 'Object three' }
```

Merge Objects

Before ES6:

```
let obj1 = { sub_one: "subject one" }
let obj2 = { sub_two: "subject two" }
Object.assign(obj1, obj2);
console.log(obj1);
```

o/p: `{ sub_one: 'subject one', sub_two: 'subject two' }`

After Es6:

```
let obj_1={sub_one:"Subject one"};
let obj_2={sub_two:"Subject two"};
let obj_3={sub_three:"Subject three"};

let final = {...obj_1,...obj_2,...obj_3};
console.log(final);

o/p: { sub_one: 'subject one', sub_two: 'subject two' }
```

Shallow Cloning:

One object can be directly assigned to another object.

disadvantage: if 1st argument data changed -> second argument data also changes

```
let obj_1 ={"p_id":111};
let obj_2 = obj_1;

console.log(obj_1); //{ p_id: 111 }
console.log(obj_2); //{ p_id: 111 }

obj_1.p_id=222;
console.log(obj_1); //{ p_id: 222 }
```

```
console.log(obj_2); //{ p_id: 222 }
```

Deep Cloning:

Achieved using spread operator "…"

Example:

```
let obj_1 ={"p_id":111};
let obj_2 = {...obj_1};

console.log(obj_1); //{ p_id: 111 }
console.log(obj_2); //{ p_id: 111 }

obj_1.p_id=222;
console.log(obj_1); //{ p_id: 222 }
console.log(obj_2); //{ p_id: 111 }
```

Date: 5-6-2020

Arrays

```
Example:
    <script>
        let products = [{
            p_id: 111,
            p_name: "p_one",
            p_cost: 10000
        }, {
            p_id: 222,
            p_name: "p_two",
            p_cost: 20000
        }, {
            p_id: 333,
            p_name: "p_three",
            p_cost: 30000
        }, {
            p_id: 444,
            p_name: "p_four",
            p_cost: 40000
        }, {
```

```
        p_id: 555,
        p_name: "p_five",
        p_cost: 50000
    }, ];

    document.write(`
        <table border="1"
        align="center"
        cellspacing="10px"
        cellpadding="10px">

        <thead>
            <tr>
                <th>S.NO</th>
                <th>P-Id</th>
                <th>P-Name</th>
                <th>P-Cost</th>
            </tr>
        </thead>
        <tbody>
        `);
    products.forEach((element, index) => {
        document.write(`<tr>
            <td>${index+1}</td>
            <td>${element.p_id}</td>
            <td>${element.p_name}</td>
            <td>${element.p_cost}</td>
            </tr>
            </tbody>`);
    })
</script>
```

S.NO	P-Id	P-Name	P-Cost
1	111	p_one	10000
2	222	p_two	20000
3	333	p_three	30000
4	444	p_four	40000
5	555	p_five	50000

Reading Json

Use http://jsonviewer.stack.hu/ to view json structure.

Date: 5-7-2020

Async Calls

```
We will make use of json-server for making async calls.
Json-server is used to develop REST API's ( GET, POST, PUT, DELETE etc..)
Json-server is light-weight server
Supports only json objects
Install json-server using: npm install –g json-server@latest
Npm present in NodeJs
Load json data in json-server using: "json-server –watch filename.json"
Test REST API using postman tool
Download and install postman

Create a json in a text file with .json extension.
Example: demo.json
{
  "products": [
    {
      "id": 1,
      "p_id": 111,
```

```
  "p_name": "p_one",
  "p_cost": 10000
},
{
  "id": 2,
  "p_id": 222,
  "p_name": "p_two",
  "p_cost": 20000
},
{
  "id": 3,
  "p_id": 333,
  "p_name": "p_three",
  "p_cost": 30000
},
{
  "id": 4,
  "p_id": 444,
  "p_name": "p_four",
  "p_cost": 40000
},
{
  "id": 5,
  "p_id": 555,
  "p_name": "p_five",
  "p_cost": 50000
}
```

```
  ]
}
```

Load the demo.json

Json-server --watch demo.json

* Default port number of json-server is 3000

Copy paste the URL in postman. Which is generated in json-server?

Eg:

GET http://localhost:3000/products

http://localhost:3000/products/1 : will display only product1 information.

POST http://localhost:3000/products and set data in body of postman and send, data will be added to demo.json (verify)

POST: Send the data to the server

PUT: Update the request

http://localhost:3000/products/id id has to be set and sent the data in body (make sure about the id we want to update). Check your json file, data will be modified.

DELETE: used to delete the record

http://localhost:3000/products/id - id has to be sent, to delete particular record. Check you json file

sorting in postman URL

- ➔ Sort based on p_cost in descending order
  http://localhost:3000/products?_sort=p_id&_order=desc
- ➔ Sort based on p_cost in ascending order
  http://localhost:3000/products?_sort=p_id&_order=asc

GreaterThan or Equal

- ➔ Greater than a particular id (_gte)
  http://localhost:3000/products?_id_gte=3

LessThan or Equal

➔ Less than a particular id (_lte)
http://localhost:3000/products?_id_lte=3

Searching (q = …) in complete body

→ http://localhost:3000/products?q=50000

Date: 8-5-2020

Async communication:

Below table provides information about how we can achieve ajax in different-different languages.

Jquery	Ajax
Angular	$http
Angular 10	httpClient
ReactJs	Axios

Example:

```
<script>
        $.ajax({
            method : "GET",
            url : "https://restcountries.eu/rest/v2/all",
            success : (posRes)=>{
                console.log(posRes);
            },
            error : (errRes)=>{
                console.log(errRes);
            }
        });
    </script>
```

→if get request success – result will be stored in posRes variable

→if get result failed – result will be stored in errRes variable

Integrate the above example with json-server. load product.json in json-server (json-server --watch products.json)

```
<script>
            $.ajax({
                method : "GET",
                url : "http://localhost:3000/products",
                success : (posRes)=>{
                    console.log(posRes);
                },
                error : (errRes)=>{
                    console.log(errRes);
                }
            })
        </script>
```

Post Data using POST request:

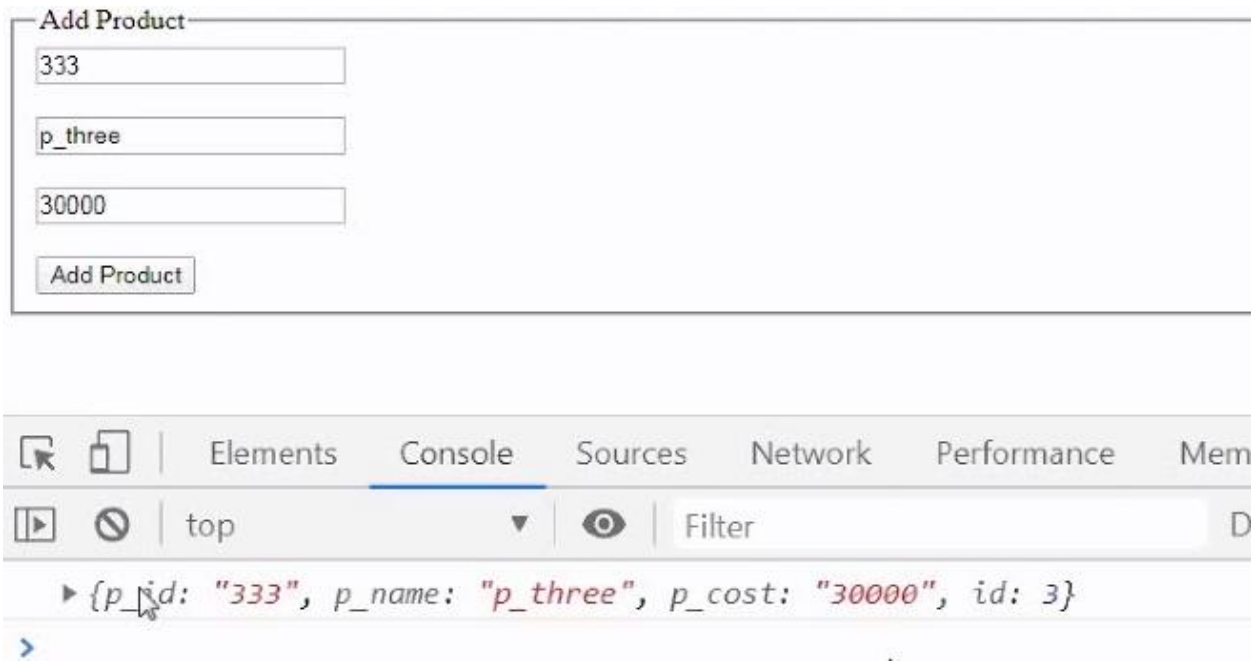
```
//field set is used to get boxed form
        <fieldset>
            <legend>Add Product</legend>

            <input type="number" id="p_id">    <br><br>
            <input type="text" id="p_name">    <br><br>
```

```
    <input type="number" id="p_cost">  <br><br>
    <button id="my_btn">Add Product</button>

</fieldset>

<script>
    $("#my_btn").click(function(){
        let record = {
            "p_id": $("#p_id").val(),
            "p_name" : $("#p_name").val(),
            "p_cost" : $("#p_cost").val()
        };
        $.ajax({
            method : "POST",
            url : "http://localhost:3000/products",
            data : record,
            success : (posRes)=>{
                console.log(posRes);
            },
            error : (errRes)=>{
                console.log(errRes);
            }
        })
    });
</script>
```



Assignment – put request and delete request.

PUT Example *(it is an assignment – not discussed in class)*

```
<script>
    let myData = {
        "id": 1500,
        "title": "App-server",
        "author": "WEBcode"
    };
    $.ajax({
        type: "PUT",
        data: myData,
        url: " http://localhost:3000/posts/" + myData.id,
        success: console.log("Success"),
    });
</script>
```

Delete Example

```
<script>
    let myData = {
        "id": 1500,
        "title": "App-server",
```

```
            "author": "WEBcode"
        };
        $.ajax({
            type: "DELETE",
            data: myData,
            url: " http://localhost:3000/posts/" + myData.id,
            success: console.log("Success"),
        });
    </script>
```

Date: 9-5-2020

Load different json file in different json-servers using different port numbers:

In terminal 1:

Json-server --watch filename1.json --port 8080

After that try URL's generated in browser or postman.

Create another terminal – Terminal 2:

Json-server --watch filename2.json --port 9090

After that try URL's generated in browser or postman.

If function need to return response 100% (either success or failure) is called as PROMISE

Asncronous calls can be made using .ajax({ })

If get request success – return positiveResponse – inform positive response to promise by using: resolve (positiveResponse)

If get request failure – return errorResponse – inform error response to promise by using: reject (errorResponse)

To consume Promise – in ES9 – two different methods are introduced – 1. Async 2. Await.

Syntax:

```
aync function functionName(){

    Let var1 = await.methodName(); // Method name is custom method name

    Console.log(var1);
```

}

From ES9 IIFE has introduced.

- ➔ Immediate Invocable Functional Expression
- ➔ Merge both declaration and calling of method
- ➔ Syntax ***(() => { }) ( );***

➔ Syntax for aync methods : ***(aync () => { }) ();***