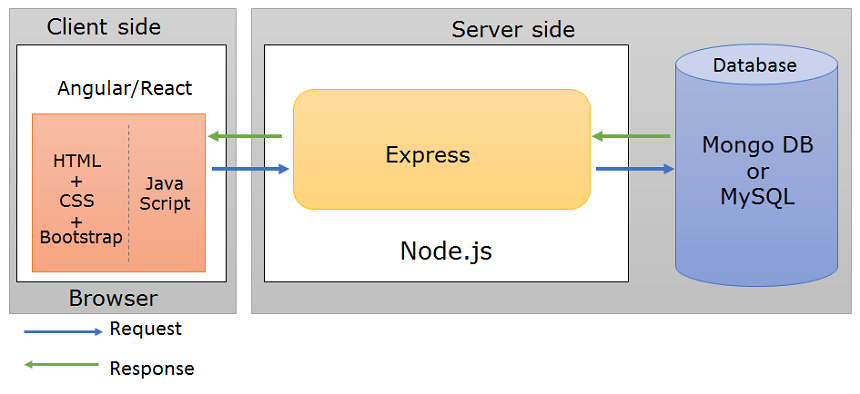
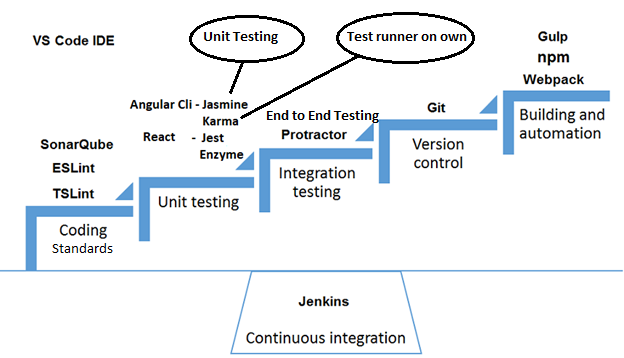
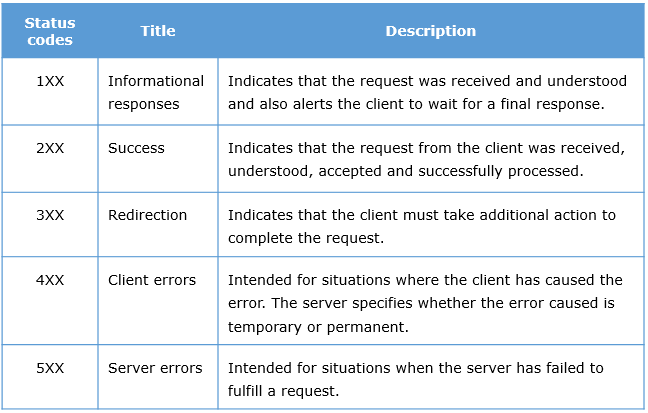
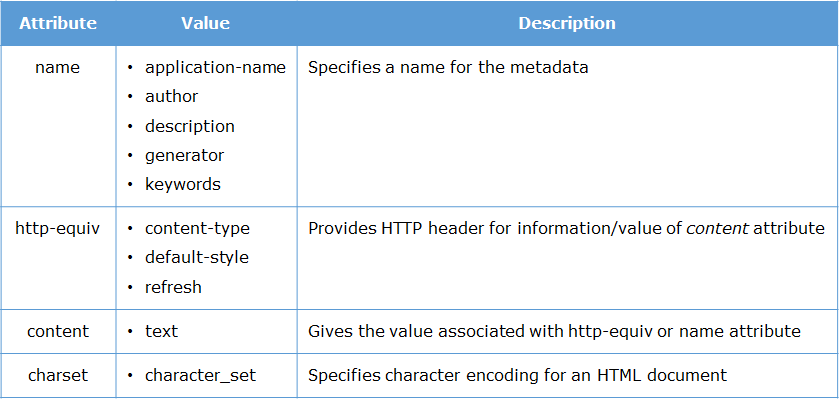
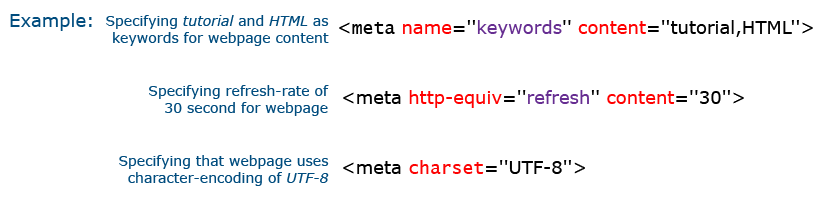
# **CSS**

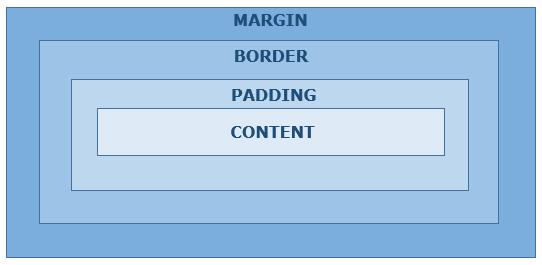


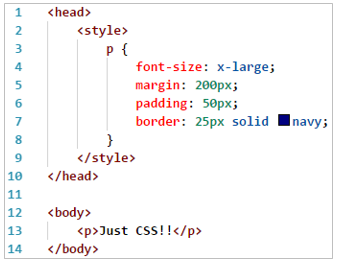


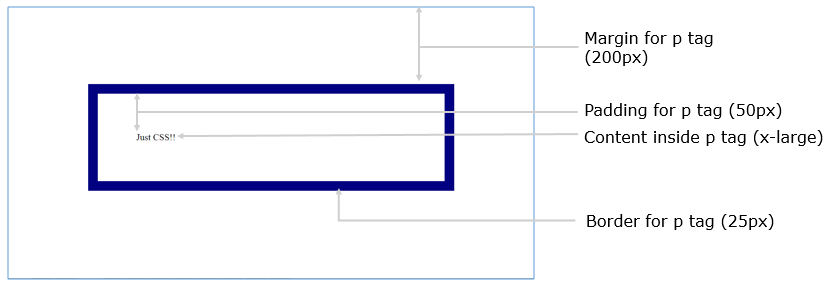












Directives used to change DOM (document object model)

structural directives - \*ngIf, \*ngFor, \*ngSwitch(A Structural directive changes the DOM structure by adding and removing DOM elements.)

attribute directives - [ngStyle],[ngClass](Attribute directives changes the appearance or behavior of a component or element.)

Animations in CSS are done using key-frames.

@keyframes exampleAnimation2 {

0% { background-color: red; }

25% { background-color: yellow; }

50% { background-color: blue; }

100% { background-color: green; }

}

div {

width: 100px;

height: 100px;

background-color: red;

animation: exampleAnimation 4s 1(iteration count) ease-in;

}

Media queries are used to specify how the web page should look on different sizes of screens

@media only screen and (max-width:500px){

body{

background-color: blue;

}

}

Syntactically Awesome Style Sheets(SASS) is considered as a CSS preprocessor and is a stylesheet language .(extension .scss)

used for declaring variables and reusing them.

$default-background-color: white;

.tour {

.tour-card {

border: 1px solid black;

background: $default-background-color;

}

}

used for nesting classes

.content-area {

display: flex;

flex-wrap: wrap;

.main {

display: flex;

flex-direction: column;

flex:8;

.tour {

display: flex;

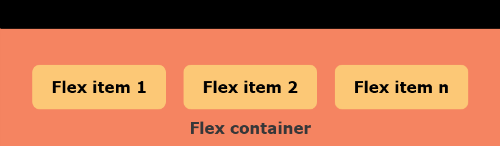
}

}

}

command to generate css file from sass

sass scss-demo.scss output.css



# **MONGO DB**

Mongodb:

db.product\_catalog.insert (

    {

        prodid:7000010,

        prodname:"nosql distilled",

        publisher:"Addison-Wesley",

        genre: {academic: "technical"},

        ISBN:1234567,

        price:400

    })

Insert acts like both insertone and InsertMany, output WriteResult({"nInserted":1})

db.product\_catalog.find({

$and:[

{manufacturer:"apple"},

{colors: { $all:["black", "silver"] } }

]},

{\_id:0, prodname:1, colors:1, price:1}

).pretty()

$all means shld contain all mentioned, $in anyone matches,

$and, $or, $not, $not

$gte, $ne

db.product\_catalog.updateMany(

{ "price" : { $gt : 80000 }, "manufacturer" : "apple" },

{ $set: { "prodname" : "iphone 7 plus" } },

{ upsert: true }

)

If upsert : true mentioned then if record not founf it will create a new record

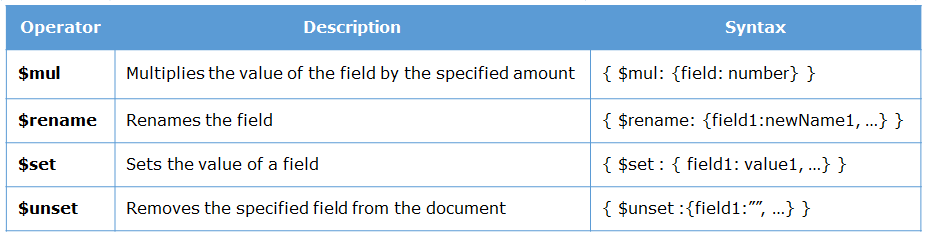
db.product\_catalog.update(

{ ISBN: 18407806 },

{ $inc: { price: 50 } }

)

$inc (+ or -), $mul (multiply or divide), $rename (change field name) , $set (sets field value), $unset (removes specified field from document)

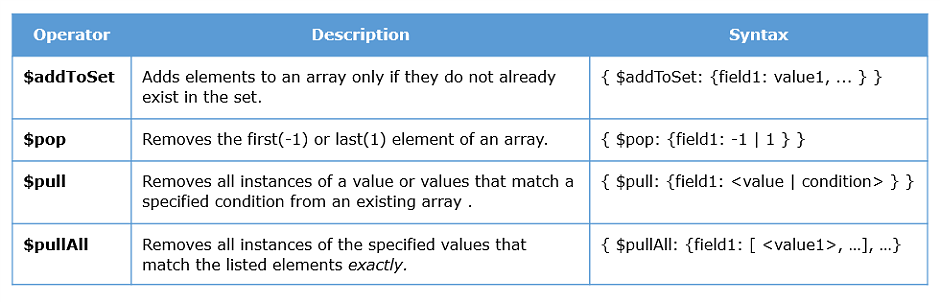


db.product\_catalog.update(

{ prodid: 7000001},

{ $push: { colors: "white" } } )

To push value into an array in db (if doesn’t exist create a new one)



Delete:

Example:

db.product\_catalog.deleteMany(

{ price: { $lt : 1000 } } )

Also has deleteOne

Aggregation :

db.product\_catalog.count( { "categories.sub": "smartphones" })

db.product\_catalog.distinct( "manufacturer" )

Example:

db.product\_catalog.aggregate( [

{ $project: { \_id: 0, manufacturer: 1, price: 1 } },

{ $group: { \_id: "$manufacturer", totalPrice: { $sum: "$price" } } }

] )

Example:(similar to find)

db.product\_catalog.aggregate(

{ $match: {$and: [{ manufacturer: "lenovo" }, {price : {$lt:10000} } ] } } )

Example:

db.product\_catalog.aggregate( [

{ $sort : { price: 1 } },

{ $limit: 5},

{ $out: "FiveCheapestMobiles"}

] )

Example:

db.product\_catalog.createIndex(

{ price:-1 })

Example:

db.product\_catalog.createIndex(

{ price:1, rating:-1 })

Example:

db.product\_catalog.createIndex(

{categories:"text"})

db.product\_catalog.find(

{ $text:{ $search:"smartphones"}})

Example:

db.product\_catalog.getIndexes()

Syntax:

If you know the name of the index

db.collection\_name.dropIndex( "index\_name" )

OR

For ascending indexes:

db.collection\_name.drop\_Index( { field\_name : 1 } )

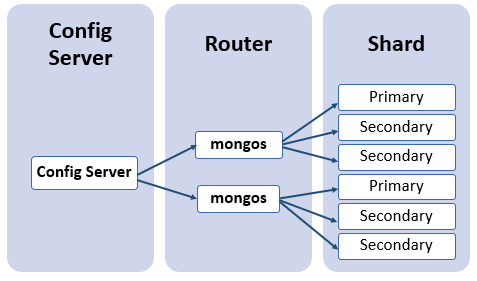
OR

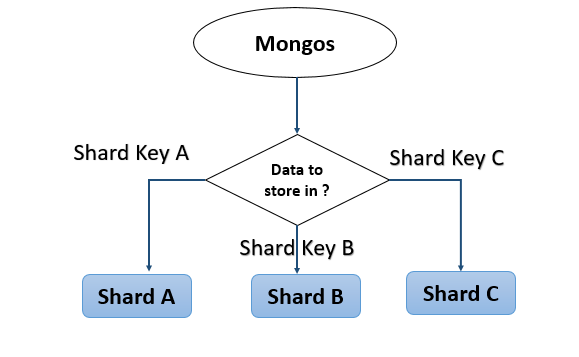
For descending indexes:

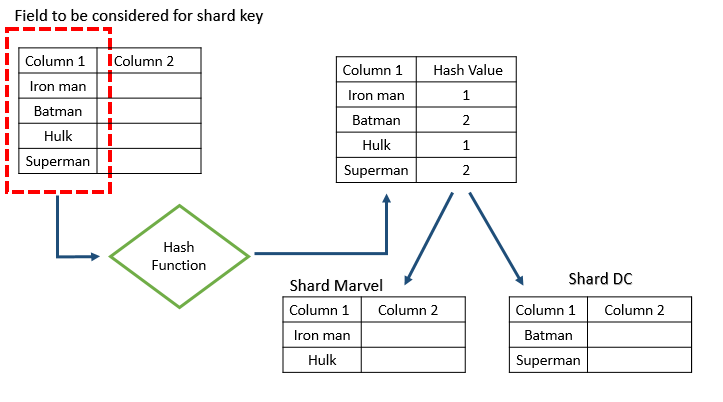
db.collection\_name.drop\_Index( { field\_name : -1 } )

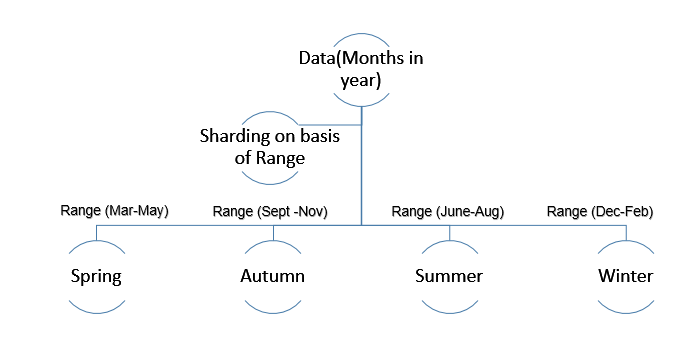
Syntax: to delete all indexes

db.collection\_name.dropIndexes()









English - is/am/are/was/were + used to + ing verb

superlative degree follows in (if it is place/group) else it follows of

eg: the shortest boy of/in the school (ans: in)

if ends with ce then it is abstract noun

present perfect/present perfect continuous we will use since not for

# **JAVASCRIPT**

* JavaScript is an interpreted language
* JS has 5 data types: number, string, undefined, null, object

# **Functions**

* Functions are objects in JS
* Functions can be stored in variables, passed as parameters or returned as values
* Functions without names are called anonymous functions
* Arrow functions are shorter way to write anonymous functions
* Variables can either be in global, local or block scope

1. Functions are actually Objects. That means a function can be stored in a Variable.
2. We can also pass Functions as a parameter to another function.
3. Before any function executes, all the Local Variables are **Hoisted**in the function. Hoisting is a phenomenon, where no matter where the variable is declared inside the function, they are all ***pushed as the first statements inside the function*** during the function execution.
4. However, only variable name is hoisted and ***not its value.***
5. The variable ***i*** has been declared with ***var*** keyword, it is ***accessible throughout the function***.
6. A ***variable with a block scope*** is accessible only within the block of statements and not throughout the function.
7. ***const*** is a keyword which is also used to create a block scoped variable. But the difference between const and let is that, a const variable ***cannot be modified***. It is Constant.

Objects have ***properties and methods***. JavaScript provides many standard **built-in** objects. In addition to that it also provides an option to create **user defined** objects.

Some of the commonly used built-in objects are:

* Array
* Date
* String

# **Array**

* .forEach()
* .map()
* .filter()
* .find()
* .push()
* .pop()
* .splice()

1. Consider the **forEach()** function of an array. This function takes another function as parameter and invokes the function for every item in the array.

placesToVisit= ["Paris", "New York", "Switzerland"];

placesToVisit.forEach(place => console.log("Trip to " + place));

// Trip to Paris

// Trip to New York

// Trip to Switzerland

1. We know that an array object has a **.map()** function that creates a new array based on what the passed callback function does.

placesToVisit= ["Paris", "New York", "Switzerland"];

placesUpperCase = placesToVisit.map(place => place.toUpperCase());

console.log(placesUpperCase);

// ["PARIS", "NEW YORK", "SWITZERLAND"]

1. We know that an array object has a **.filter()** function that returns a filtered sub array based on what the passed callback function does.

placesToVisit = ["Paris", "New York", "Switzerland"];

filteredPlace = placesToVisit.filter(place => place.length > 5);

console.log(filteredPlace);

// [ 'New York', 'Switzerland' ]

1. We know that an array object has a **.find()** function that returns the first element in the array based on what is passed as callback function.

placesToVisit = ["Paris", "New York", "Switzerland"];

findPlace = placesToVisit.find(place => place.length > 5);

console.log(findPlace);

// "New York"

**Palindrome**

function isPalindrome(s)

{

var reversedText = s.toLowerCase().split('').reverse().join('');

return s=== reversedText;

}

console.log(isPalindrome("level"));

# Objects

Creating **Object** using object literal

var empOne = {

name : "John",

empNumber : 1001,

emailId : "John@gmail.com",

swipeIn(){console.log("Swipe In by "+this.name)}

};

**Accessing object properties**

object.property

object[property]

**Iterating an object**

for..in

object.values(ObjectName)

Using Object Destructuring we can destructure an existing object into variables. If the var is prefixed by 3 dots, then it is a Rest Variable and can store more than one property.

# Class

