Java Script Essentials:

1. Spread operator: whenever you want to concatenate a string or convert a object which is a set to a list or when you want don't want to change initial along with the later array, use spread operator.
   Ex: arr1 = [1,2,3,4], arr2 = arr1, arr2.push(5)
   When we do console log both arr1 and arr2 will have 5, and hence we come up with spread operator.
   That is, arr2 = […arr1], which is not a copy but spreading the operators in arr1 to array2.

2. Arrow function:
   Function name(){
   Return "shivani";
   }
   Var name;

   When written as
   Var name = Function(){
   Return "shivani";
   }

   It becomes anonymous function, but when written as

   Var name = () => {

   Return "shivani";

   }

   Becomes arrow function.

   Arrow function in reduced form is:

   Var name = () => "shivani";

3. Map functions:

   Consider an object

```
const people = [
  {
    name:'bob',
    age:20,
    position:'developer',
  },
  {
    name:'anna',
    age:25,
    position:'designer',
  },
  {
    name:'susy',
    age:27,
    position:'boss',
  }
];
```

In order to retrieve their values individually and keeping for reference we use map function.

```
const ages = people.map((ages) => {
  return ages.age*10;
});
```

This modifiys existing ages with *10.

4. Unique Operator:

```
 set const people=[
  {
    name:'bob',
    age:20,
    position:'developer',
  },
  {
    name:'anna',
    age:25,
    position:'designer',
  },
  {
    name:'susy',
    age:27,
    position:'boss',
  },
  {
    name:'shivani',
    age:20,
    position:'developer',
  },
  {
    name:'cena',
    age:20,
    position:'manager',
  },
  {
    name:'john',
    age:20,
    position:'manager',
  },
];

const ages = ["all",...new Set(people.map((ages) => {
  return ages.position;
}))];

const res = document.querySelector('#result');
res.innerHTML = ages.map((categories) =>{
 return `<button>${categories}</button>`;
}).join(" ");
```

5. Dynamic Objects:

```
const file = {
    name: "john",
    lastname: "",
    age:18
};
states('lastname','cena');
console.log(file);
{name: 'john', lastname: 'cena', age: 18}

var key = 'computer';
const file1 = {
    [key]: "john",
    lastname: "",
    age:18
};
console.log(file1);
{computer: 'john', lastname: '', age: 18}
```

6.
```
  const ages = people.filter((person) =>{
    return person.age > 30;
})
```

7. <u>Find</u>

```
 const inventory = [
 {name: 'apples', quantity: 2},
 {name: 'bananas', quantity: 0},
 {name: 'cherries', quantity: 5}
];
const val = inventory.find((person) => {
   return person.name == 'apples'
});
```

8. Reduce:

```
const ages = people.reduce((total,person) =>{
   total += person.age;
   return total;
},0);
```

   Here in reduce, we need to either reduce it to an object, number, or array, the zero mentioned above indicates that I have converted it into a number that is the initial value.
   Total is acc and person is curr it is current iteration value.
   And DON'T FOREGET TO GIVE RETURN………. if u don't u get error.

   [here it return zero at start, as we have put it to zero]

9. const people=[

```
 {
   name:'bob',
   age:20,
   position:1,
 },
 { age:27.7,
   position:2,
 },
 {name:'shivani',
   age:20.8,
   position:2,
 },
];
let {totalNumber, totalAmount} = people.reduce((total,person) => {
  let {age,position} = person
  total.totalNumber += age;
  total.totalAmount += age*position;
  return total;
},{
  totalNumber : 20,
  totalAmount :0,
});
console.log(totalAmount);
//converting it it no ending with 2 decimal and converting it into float
totalAmount = parseFloat(totalAmount.toFixed(2));
console.log(totalAmount);
```

10. Destructing array

```
const arr = ['shivani','radha','tg','shivappa'];
const [gowda, br, gowdan, ks] = arr;
console.log(gowda);
```

11. Destructuring array

```
const arr = ['gowda','shivani'];
const [second, first] = arr;
console.log(first+" "+second);
```

12. Destructuring objects
    use flower brackets instead {}

```
const people=
  {
    name:'bob',
    age:20,
    position:1,
  }
let {age, position} = people;
console.log(age)
console.log(position)
```

13. Rest operator

Arrays

```
const arr = ['gowda','shivani',"shivappa"];

const [name,...rest] = arr

console.log(name);

console.log(rest);
```

on objects

```
const people=
  {
    name:'bob',
    age:20,
    position:1,
    time: 9,
  }
let {age,…rest} = people;
console.log(age)      //20
console.log(rest)    //  {name:'bob', position:1,time: 9}
```

function

```
const getAvergae  = (name,…scores) => {

            console.log(name); // bob (after invoking function)
            console.log(scores); //null
}

getAverage(person.name)
```

(now suppose I start adding numbers like getAvergae(person.name,90,80,70,20,19,18);
the array starting from 90 to 18 will end getting attached at scores.)
so now console.log(scores);   //90,80,70,20,19,18

14. Spread operator

```
const arr = "shivani";
const letter = [...arr];
console.log(letter); //['s','h','i','v','a','n','i']

const arr = ["shivani", "gowda"];
const arr2 = ["radha", "shivappa"];
const arr3 = [arr, arr2]
```

```
console.log(arr3)    //[ [ 'shivani', 'gowda' ], [ 'radha', 'shivappa' ] ]


const arr3 = [...arr, ...arr2]
```

```
[ 'shivani', 'gowda', 'radha', 'shivappa' ]
```

<u>On obejcts</u>

```
Const person = {name:’john’,job:’developer’};
Const newPerson = {…person};
```

15. <u>Call Back function</u>

```
function ctUpper(value){

        console.log(value.toUpperCase())
       }

function handelName(name,ch){

        const fullName = name + " gowda";
        ch(fullName);
       }

// we are not calling ctUpper here, but only handelName  and hence we dont

       // ctUpper()
       handelName('shivani',ctUpper)
```

------------------------*--------------------------------------*------------------------------------*-------------------------------

```
       function handelName(name,ch){
        const fullName = name + " gowda";
        ch(fullName);
       }

       // we are not calling ctUpper here, but only handelName  and hence we dont
       // ctUpper()
       handelName('shivani',function(value){
        console.log(value.toUpperCase());
       })
```

16. This is Call back hell

```
       const first = document.querySelector("#first");
        const second = document.querySelector("#second");
        const third = document.querySelector("#third");

        const btn = document.querySelector("#btn");

        btn.addEventListener('click',() => {
          setTimeout(() =>{
```

```
         first.style.color ='red'    //after 5s this will start getting executed along with the below line
             setTimeout(() =>{
               second.style.color ='blue' //after next  5s
                  setTimeout(() =>{
                     third.style.color ='green' //after next to next 5s
                  },5000)
                },5000)
           },5000)
        })
```

17. Promise

```
const val = 2

const promise = new Promise((resolve,reject) => {

const random = Math.floor(Math.random * 3)

  if (random === val){

    resolve('you guessed it right')

  } else{

    reject('wrong guess')

  }

})

promise.then((data) => console.log(data)).catch((err) => conole.log(err));
```

18.

```
const btn = document.querySelector("#btn");

 btn.addEventListener('click',() => {

      addColor(2000,'#first','red','hello world')

      .then((data) => addColor(3000,'#second','green',data))

      .then((data) =>{

        console.log(data)

        addColor(4000,'#third','blue',data))

      }

      .catch((err) => console.log(err));

  })
 function addColor(time,selector,color,data){
   const element = document.querySelector(selector)
   return new Promise((resolve,reject) =>{
     if(element){
       setTimeout(() =>{
         element.style.color = color;
         resolve(data) //if u dont have resolve here
//then function will think that it is still pending and not resolved
       },time)

     }else{
       reject(`the "${selector}" is not found`)
     }
   })
 }
 19 const users = [
 { id : 1, name:"John"},
 { id : 2, name:"susan"},
 { id : 3, name:"bob"},
]

const articales = [
 { userId : 1, articles: ['one','two','three']},
 { userId : 2, articles: ['four','five']},
 { userId : 3, articles: ['six','seven','eight']},
]

const getData = async() => {
    const user = await getUser('John')
    if(user){
      const articles = await getArticles(user.id)
      console.log(articles)
    }
}

getData()

// getUser("John").then((data) => {console.log(data)})
//              .then((articles) => {console.log(articles)})
//              .catch((data) => {console.log(data)})

function getUser(name){
  return new Promise((resolve,reject) => {
    const user = users.find((user) => user.name === name)

    if(user){
      resolve(user)
    }else{
      reject("Sorry, user name not found!")
    }
  })
}

function getArticle(UserId){
  return new Promise((resolve,reject) => {
    const userArticles = articles.find((user) => user.userId === userId)
    if(userArticles){
      return resolve(userArticles.articles)
    }else{
      reject("wrong ID")
    }
  })
}
```

20.

```
const url = 'https://www.course-api.com/react-tours-project'
fetch(url)
.then((res) => re.json())
.then((data) => console.log(data) )
.catch((err) => console.log(err))
```