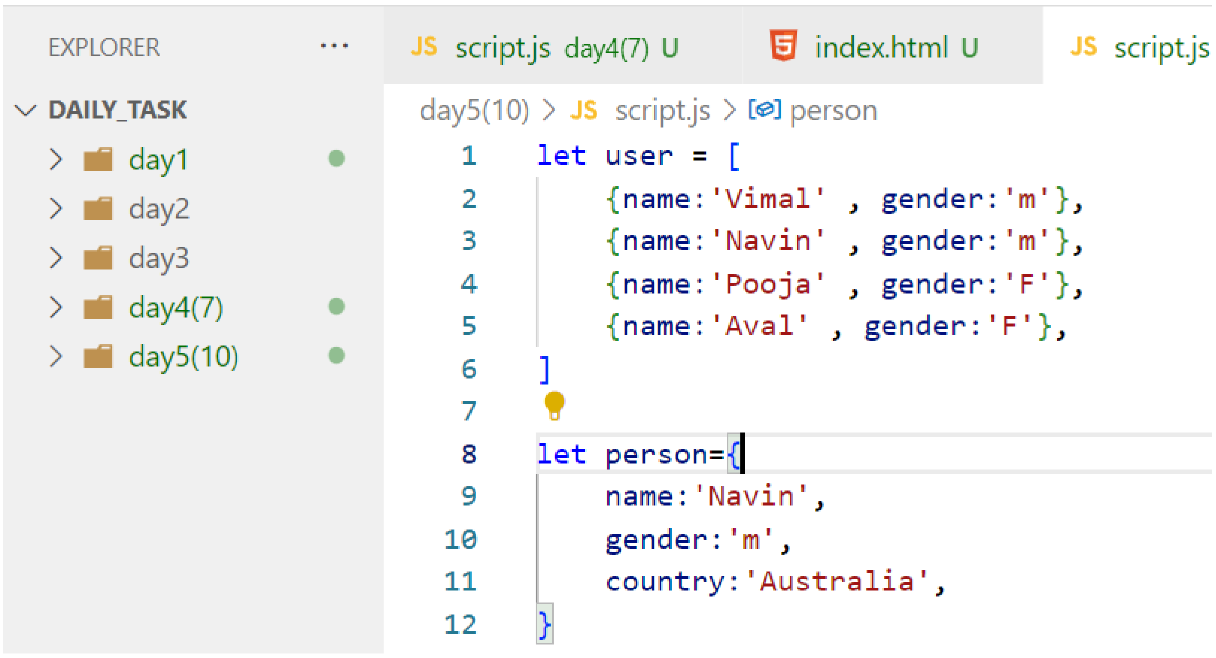
Day - 5

10.02.2025



```
let user = [
    {name:'Vimal' , gender:'m'},
    {name:'Navin' , gender:'m'},
    {name:'Pooja' , gender:'F'},
    {name:'Aval' , gender:'F'},
]

let person={
    name:'Navin',
    gender:'m',
    country:'Australia',
}
```

For loop ;

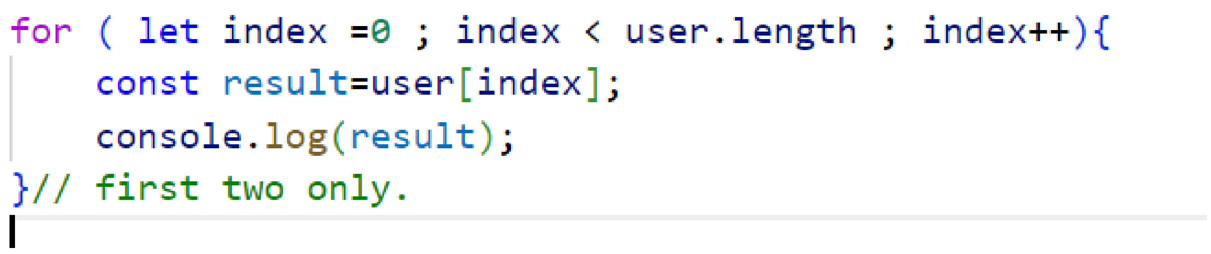
For loop - Loops through a block of code a number of times

Expression 1 sets a variable before the loop starts (let i = 0).

Expression 2 defines the condition for the loop to run (i must be less than 5).

Expression 3 increases a value (i++) each time the code block in the loop has been executed.

```
for ( let index =0 ; index < user.length ; index++){
        const result=user[index];
     console.log(result);
}// first two only. Break use
```



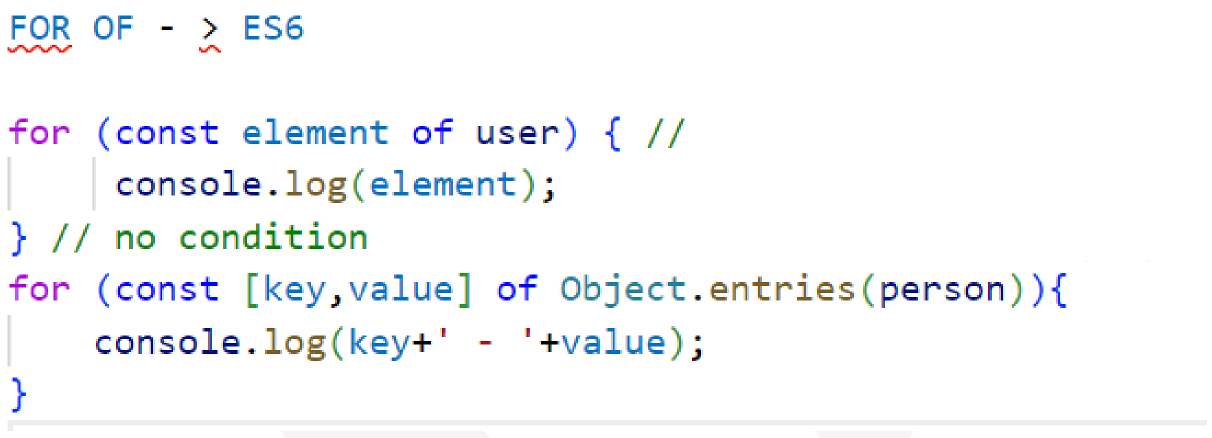
```
for ( let index =0 ; index < user.length ; index++){
    const result=user[index];
    console.log(result);
}// first two only.
```

For of loop ;

The JavaScript for of statement loops through the values of an iterable object.

It lets you loop over iterable data structures such as Arrays, Strings, Maps, NodeLists

```
for (const element of user) { //
    console.log(element);
} // no condition
for (const [key,value] of Object.entries(person)){
   console.log(key+' - '+value);  }
```



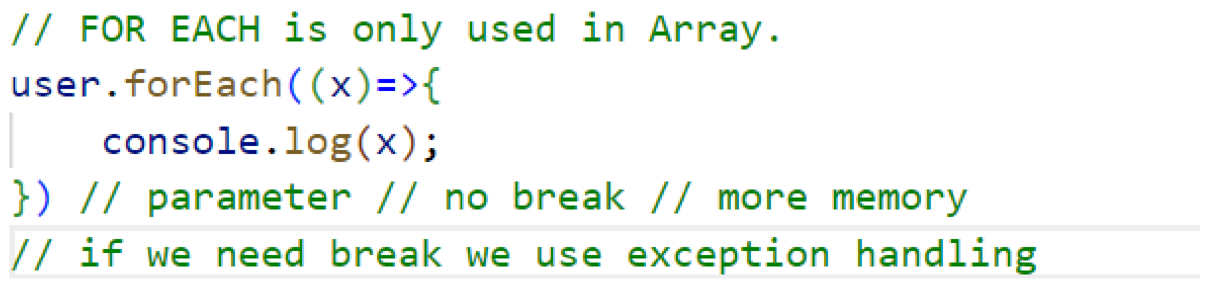
```
FOR OF - > ES6

for (const element of user) { //
        console.log(element);
} // no condition
for (const [key,value] of Object.entries(person)){
      console.log(key+' - '+value);
}
```

For each loop:

The forEach() method calls a function for each element in an array.

The forEach() method is not executed for empty elements.

```
// FOR EACH is only used in Array.
user.forEach((x)=>{
   console.log(x);
}) // parameter // no break // more memory
// if we need break we use exception handling
```



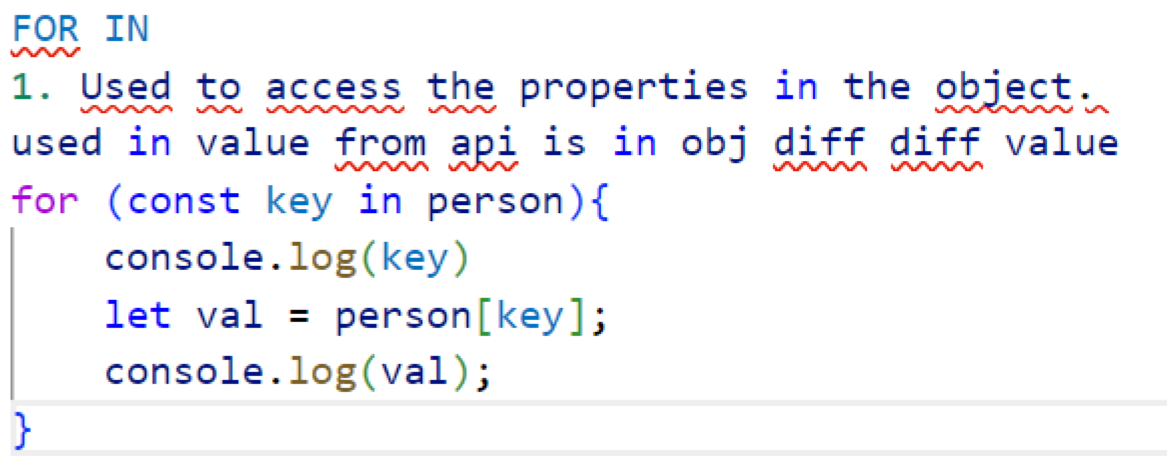
```
// FOR EACH is only used in Array.
user.forEach((x)=>{
      console.log(x);
}) // parameter // no break // more memory
// if we need break we use exception handling
```

For In loop:-

- The for in loop iterates over a person object
- Each iteration returns a key (x)
- The key is used to access the value of the key
- The value of the key is person[x]

1. Used to access the properties in the object.
2. Used in value from api is in obj diff diff value

```
for (const key in person){
   console.log(key)
   let val = person[key];
   console.log(val);
}
```

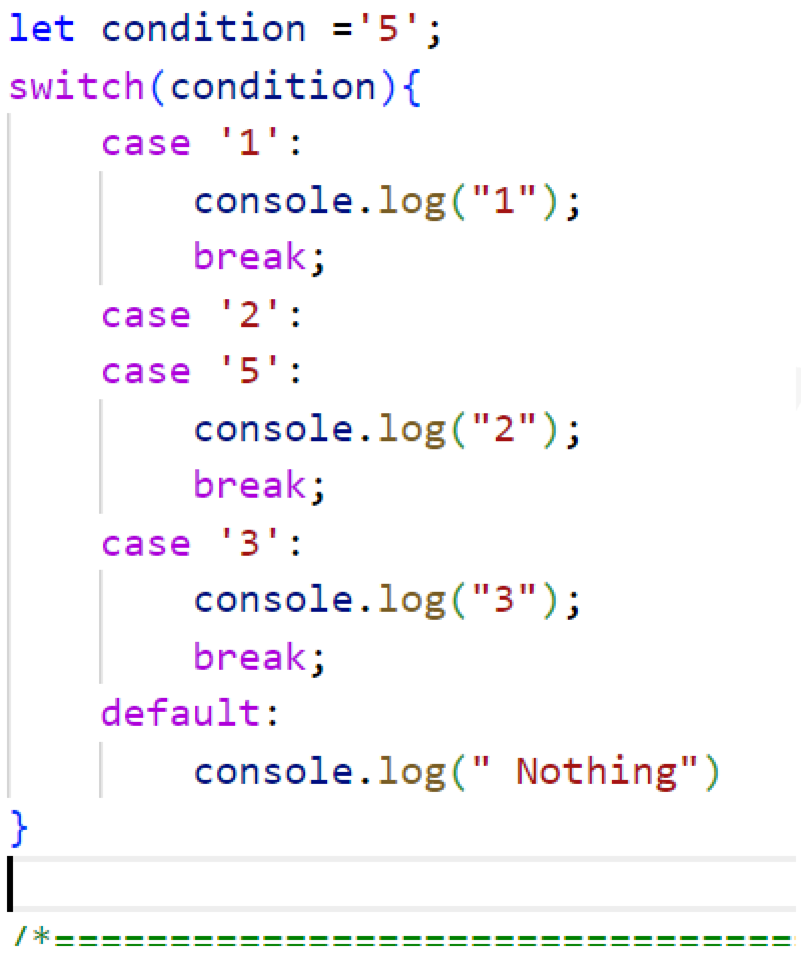


```
FOR IN
1. Used to access the properties in the object.
used in value from api is in obj diff diff value
for (const key in person){
    console.log(key)
    let val = person[key];
    console.log(val);
}
```

Switch case :-

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- If there is no match, the default code block is executed.

```
let condition ='5';
switch(condition){
    case '1':
        console.log("1");
        break;
    case '2':
    case '5':
        console.log("2");
        break;
    case '3':
        console.log("3");
        break;
    default:
        console.log(" Nothing")
}
```



```
let condition ='5';
switch(condition){
    case '1':
        console.log("1");
        break;
    case '2':
    case '5':
        console.log("2");
        break;
    case '3':
        console.log("3");
        break;
    default:
        console.log(" Nothing")
}

/*==================================
```

Async and Await:-

The async function declaration creates a binding of a new async function to a given name. The await keyword is permitted within the function body, enabling asynchronous, promise-based behavior to be written in a cleaner style and avoiding the need to explicitly configure promise chains.

```
Async -> Always return a promise
async function fn(){
   return 'Hello';
}
console.log(fn());
fn().then(val=>console.log(val))
think that are inside the then (resolve method)
let reach = new Promise((resolve, reject)=>{
   const reached = false;
   if(reached){
      setTimeout(resolve,3000,"vidhya reached");
   }else{
      reject("vidhya not reach");
}})
async function stauts(){
   try{
   console.log('hi ... ')
   let result=await reach
   console.log(result)
   console.log('bye')
   }
   catch(err){
      console.log(err)
   }
}
stauts()
```

Methods :-

A function inside an object is called methods.

```
const person1 = {
   fname: 'Naveen',  // property
   lname: 'Prakash',
   age: 22,
   fullname: function(){ //methods
     //return this.fname+' '+this.lname;
     return `${this.fname} ${this.lname}`;
   }
}
// getting obj value .. calling obj
document.getElementById("res").innerHTML=  person1.fullname() // person1.fname
// calling method
person1.fullname();
```

Promise:-

A Promise is a proxy for a value not necessarily known when the promise is created. It allows you to associate handlers with an asynchronous action's eventual success value or failure reason. This lets asynchronous methods return values like synchronous methods: instead of immediately returning the final value, the asynchronous method returns a promise to supply the value at some point in the future.

```
let reachA = new Promise ((resolve,reject)=>{
   let A=true
   if(A){
      setTimeout(resolve,3000,"A reached");
   }else{
      reject("A not reached")
   }
})

let reachB = new Promise ((resolve,reject)=>{
    let B=false
    if(B){
        setTimeout(resolve,10000,"B reached");
    }else{
        reject("B not reached")
    }
})

let reachC = new Promise ((resolve,reject)=>{
    let C=true
    if(C){
        setTimeout(resolve,1000,"C reached");
    }else{
        reject("C not reached")
    }
})

let reachD = new Promise ((resolve,reject)=>{
    let D=false
    if(D){
        setTimeout(resolve,4000,"    D reached");
    }else{
        reject("D not reached")
    }
})

Promise.all([reachA, reachB, reachC, reachD]).then((msg=>console.log(msg)))
.catch((msg)=>console.log(msg));
```

Promise.all - >

All True - Print all true.

Any False - Flow first False printed.

Promise.Any - >

All True - Second first True.

Any False - Second first True.

Promise.Race - >

All True - Second first True.

Any False - Flow first False printed.