JS

- Compiler nahi interpreter astay.
- Var ,const , let
    - **-- Difference between var and let**

1. **Javascript has main2 versions**

   Es5 (var)

   Es6 (let and const)

2.
    - **Var function scoped hota hai(var apne parent function me kahi bhi use ho sakta hai)**

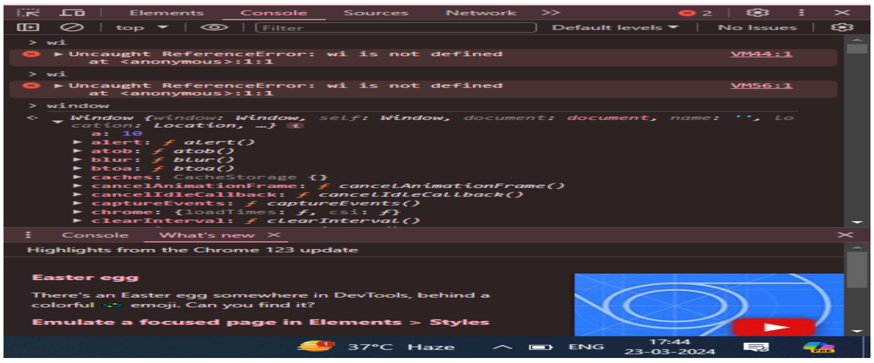
```
function hello(){
    for(var i=0;i<6;i++){
        console.log(i)
    }
    console.log(i)
}
hello()
```

0 1 2 3 4 5 aani 6 la loop terminate zal aani baher yeun 6 print zal(var is function scoped)

- **Let braces scoped hota hai**
  Above example if we use let it will not allowed to use i out of braces of for loop it will throw error

3. **Var adds itself to the window object**
   Let does't adds

```
var a=10;
let b=20;
```



```
> wi
✖ ▸Uncaught ReferenceError: wi is not defined                VM44:1
      at <anonymous>:1:1
> wi
✖ ▸Uncaught ReferenceError: wi is not defined                VM56:1
      at <anonymous>:1:1
> window
<- ▾Window {window: Window, self: Window, document: document, name: '', lo
    cation: Location, …}
      a: 10
    ▸ alert: ƒ alert()
    ▸ atob: ƒ atob()
    ▸ blur: ƒ blur()
    ▸ btoa: ƒ btoa()
    ▸ caches: CacheStorage {}
    ▸ cancelAnimationFrame: ƒ cancelAnimationFrame()
    ▸ cancelIdleCallback: ƒ cancelIdleCallback()
    ▸ captureEvents: ƒ captureEvents()
    ▸ chrome: {loadTimes: ƒ, csi: ƒ}
    ▸ clearInterval: ƒ clearInterval()
```

Highlights from the Chrome 123 update

Easter egg

There's an Easter egg somewhere in DevTools, behind a colorful emoji. Can you find it?

Emulate a focused page in Elements > Styles

a is added to the window and b does not added to the window

- JS mein kuch features nahi hai jo hum use karte hai jo ki window ke hai(alert,prompt,console,document)

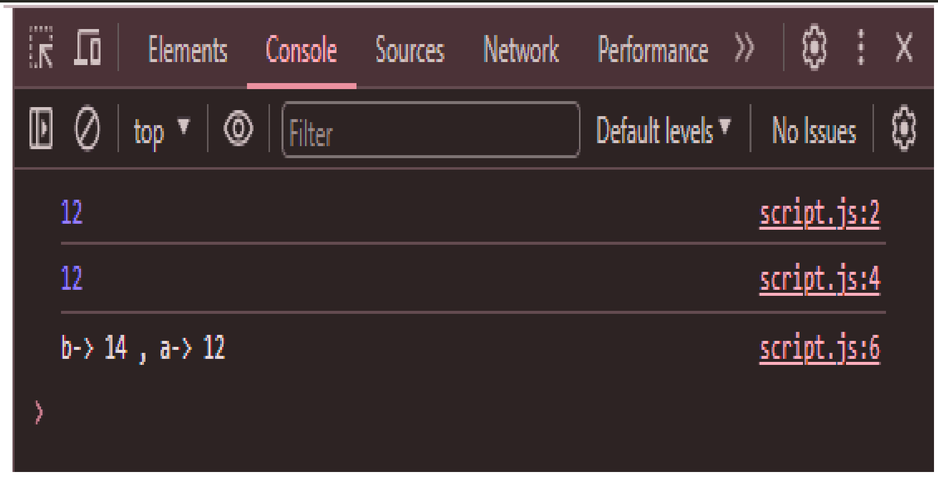
- Hoisting in js - variables declare karnya aadhich aapan tyanna use karu shakto
  - console.log(a)
  - Var a=12;

How it happens ki variable declaration top la jat
It looks in bts

Var a
console.log(a)
a=12;

- In js if variable is not initialized or fakt declare kelela aahe value assign nahi keli tr value kahi tari aste tyachi ti manje **undefined.**
- **Primitives and reference in js**
  - Primitives : string , number , null ,boolean , undefined
  - reference : [ ] ( ) { }
  - Aisi koi bhi value jisko copy karne par real copy nahi hota, balki us main value ka reference pass hojaata hai, use hum reference kehte hai, aur jiska copy karne par actual/real copy ho jaaye wo primitive hota hai.
  - Primitive

```
var a=12
console.log(a)
var b=a;
console.log(b)
b=14;
console.log("b-> "+b+" , "+"a-> "+a)
```



```
12                                   script.js:2
12                                   script.js:4
b-> 14 , a-> 12                      script.js:6
```

  - Reference

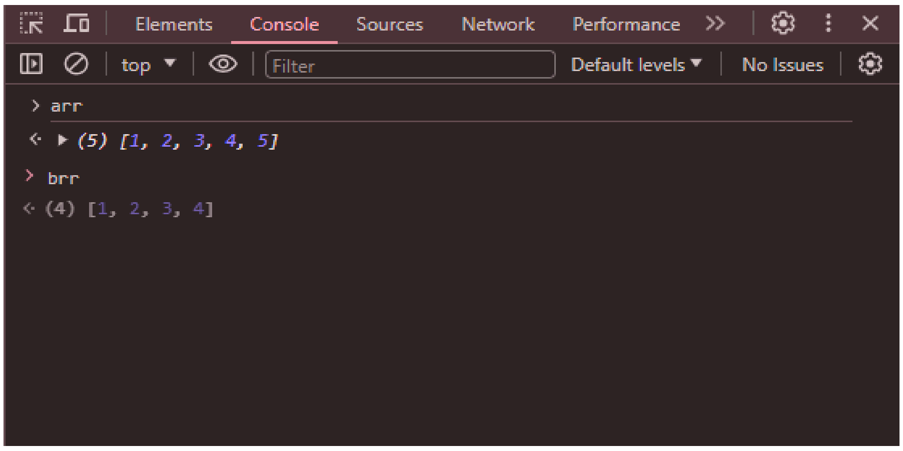
```
var arr=[1,2,3,4,5]
var brr=arr
brr.pop()
```

- A madhun pn pop() hotat values

- **- - how to copy reference values in js**

```
var arr=[1,2,3,4,5]
var brr=[...arr]
brr.pop()
```

Atta b madhe copy janar a chi naki reference



```
> arr
< ▶ (5) [1, 2, 3, 4, 5]
> brr
< (4) [1, 2, 3, 4]
```

- If else ,else if

```
if(10>9){
    console.log(10)
}else if(10<9){
    console.log(9)
}else{
    console.log(-1)
}
```

- **- - truthy vs falsy**
Falsy values : 0 , false , undefined , null , NaN , document.all
Truthy values all rest of values are

```
if(-1){
    console.log("hey")
}else{
    console.log("hello")
}
```

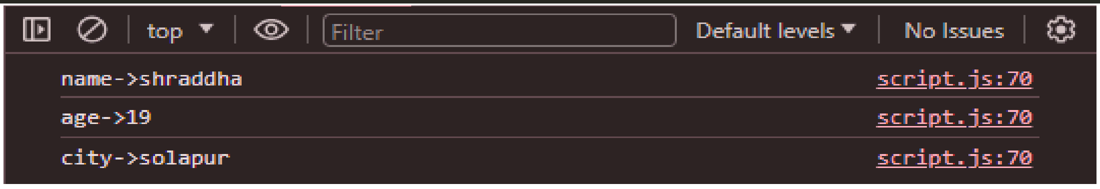
Hey

```
if(0){
    console.log("hey")
}else{
    console.log("hello")
}
```
hello

- Loops
    - For
    ```
    for(var i=0;i<10;i++){
        console.log(i)
    }
    ```
    - While
    ```
    var j=1;
    while(j<=5){
        console.log(j)
        j++;
    }
    ```

- **- - Foreach , forin**

Foreach: sirf arrayy pe chalta hai
```
var arr=[1,2,3,4,500]
arr.forEach(function(val) {
    console.log(val)
});
```

Forin : loop on objects
```
var obj={
    name:"shraddha",
    age:19,
    city:"solapur"
}
for (const key in obj) {
    console.log(key+"->"+obj[key])
}
```


```
name->shraddha                                          script.js:70
age->19                                                 script.js:70
city->solapur                                           script.js:70
```

- **Functions**
  ```
  function hello(){
      console.log("Hello world!")
  }
  hello()
  ```
  - **Arguments** : real value jo hum pass karte hai function calling ke vakt
  - **Parameters** : variables jinme value store hoti hai arguments wali.

- Array
  ```
  var array=[1,2,3];
  console.log(array[1])
  ```
  - >>2
  - 0 based indexing

- Push pop shift unshift
  ```
  var array=[1,2,3];
  // console.log(array[1])
  array.push(5)//add at last
  array.pop()//delete last
  array.unshift(8)//add at last
  array.shift()//delete front

  console.log(array)

  ```

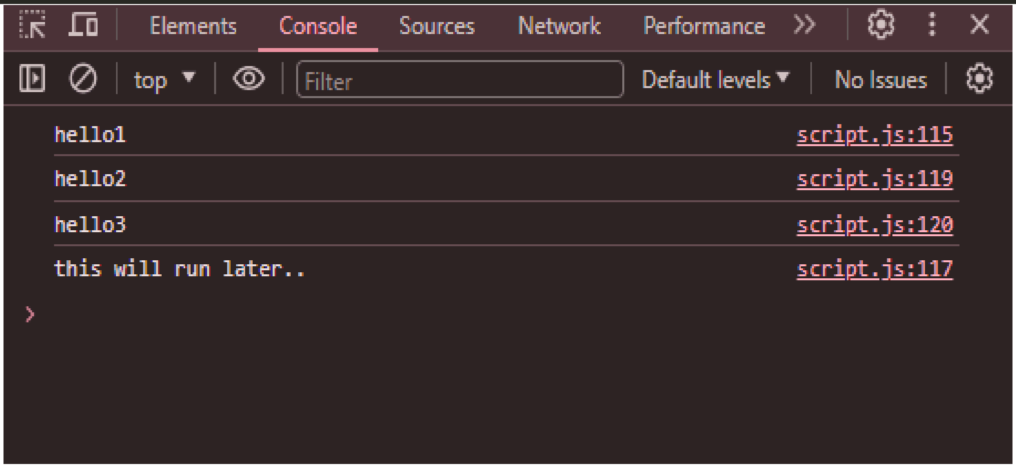
- **Execution Context**
  - **Execution context matlab jab bhi hum function chalaayenge tab function apna ek khudka ek imaginary container bana lega jisme uski cheeje hogi**
  - **1. Variables**
  - **2. Functions inside that parent function**
  - **3. Lexical environment of that function*(perticular func (scope)kin chizo ko access kr sakte hai or kise nahi)**
  - **Ye ek imaginarydibba/ container hai isse hi hum exicuton context kehte hai**
- **Objects**
```
var obj={name:"shraddha"};
console.log(obj)
```

Callbacks, Promises & Async-await

Asynchronous function

```
setTimeout(function(){
    console.log("5 sec delay");
},5000)
```

```
console.log("hello1");
setTimeout(() => {
    console.log("this will run later..")
}, 5000);
console.log("hello2");
console.log("hello3");
```



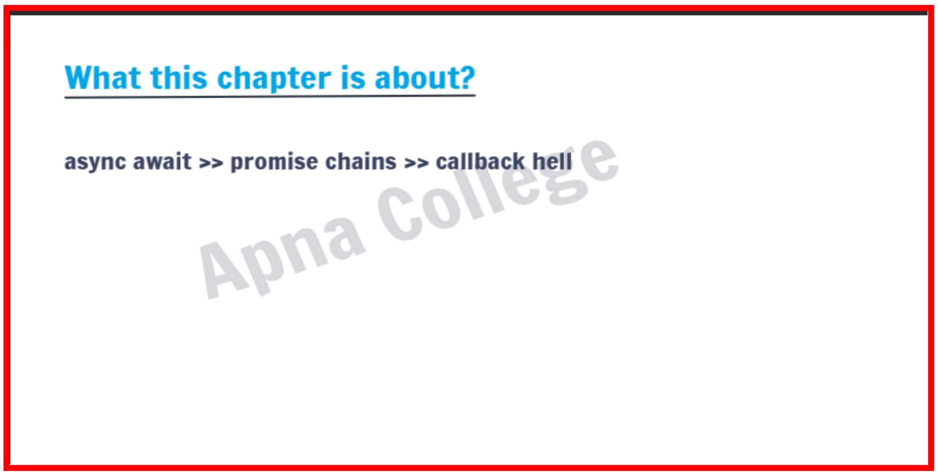
```
hello1                                             script.js:115
hello2                                             script.js:119
hello3                                             script.js:120
this will run later..                              script.js:117
>
```

Arrow function

```
setTimeout(()=>{
    console.log("5 sec delay");
},5000)
```

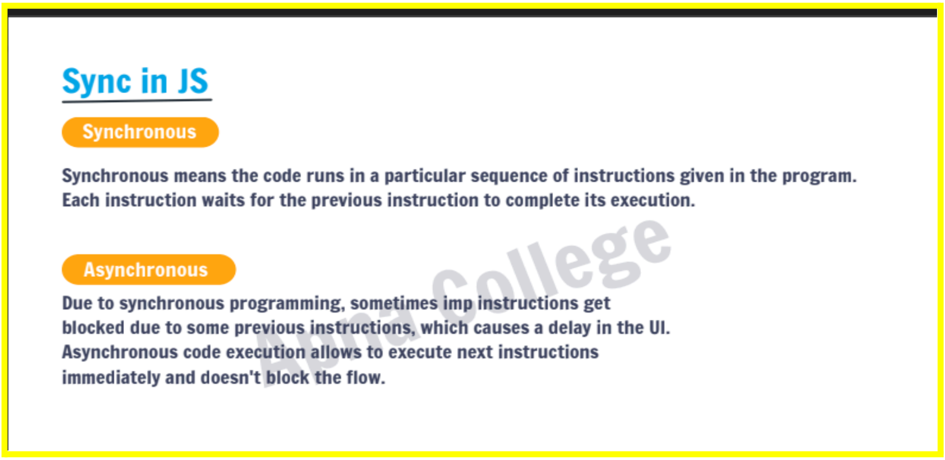
Callback function

```
function sum(){
    console.log("sum")
}
function calc(num1,num2,callbacksum){

    console.log(num1+num2);
    callbacksum()
}
calc(2,3,sum)
```



What this chapter is about?

async await >> promise chains >> callback hell



Sync in JS

Synchronous

Synchronous means the code runs in a particular sequence of instructions given in the program. Each instruction waits for the previous instruction to complete its execution.

Asynchronous

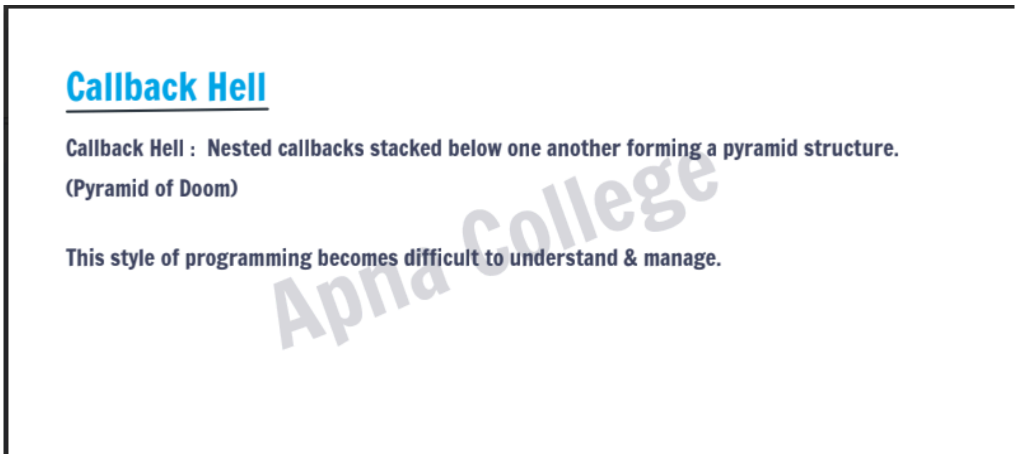
Due to synchronous programming, sometimes imp instructions get blocked due to some previous instructions, which causes a delay in the UI. Asynchronous code execution allows to execute next instructions immediately and doesn't block the flow.

Callback : function ke andar dusra function pass karto tyala callback mantat. Callbacks synchronous kam kartat aani with setTimeout function ni asynchronous ni kam kartat

Callback hell

Window+.(dot)

Nesting of callback functions.



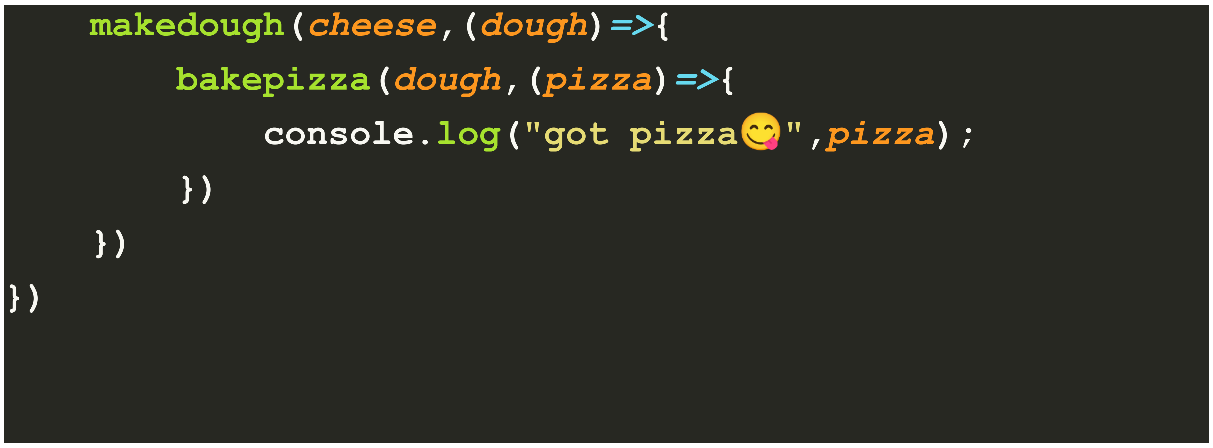
Callback Hell

Callback Hell : Nested callbacks stacked below one another forming a pyramid structure.

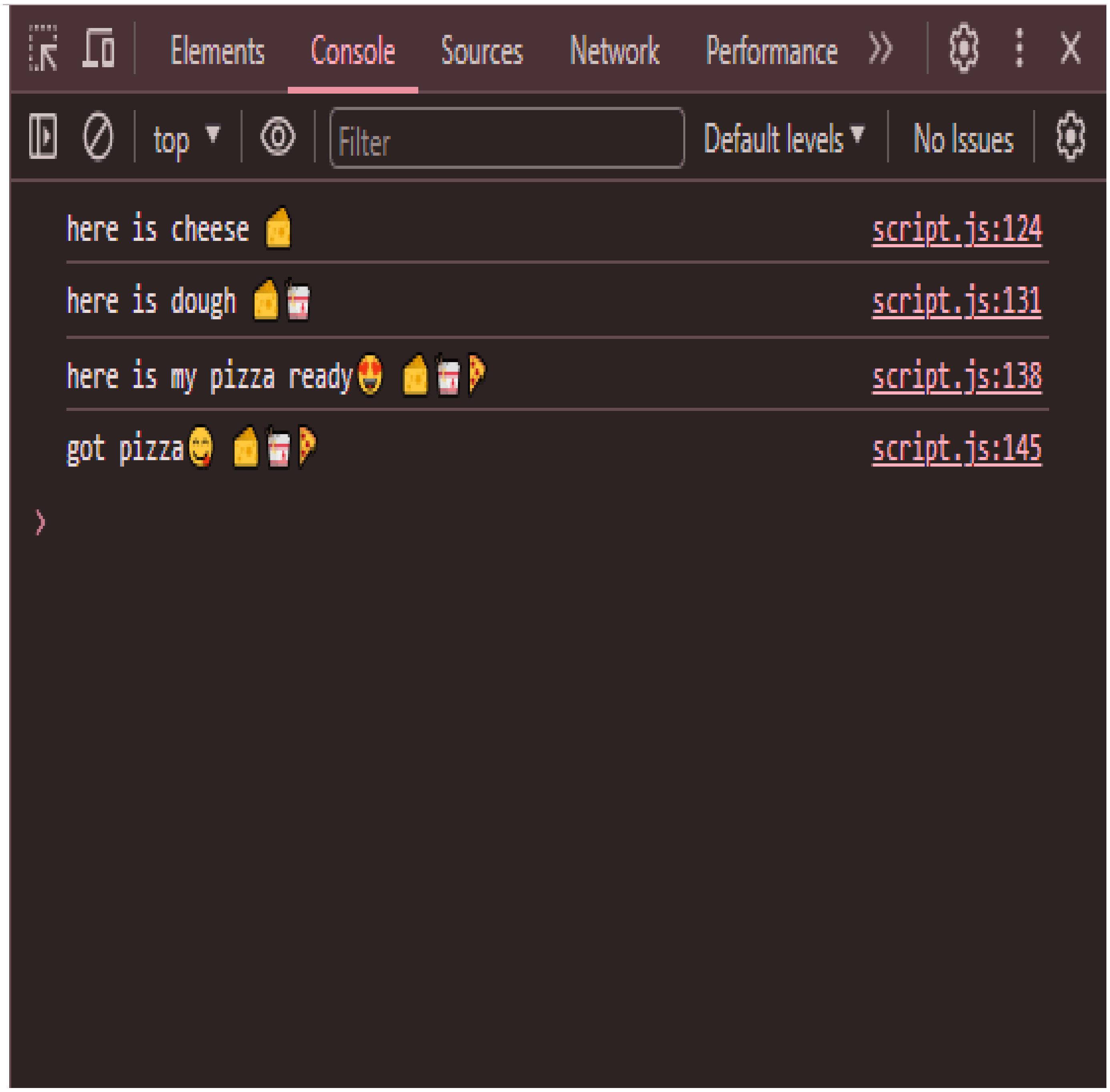
(Pyramid of Doom)

This style of programming becomes difficult to understand & manage.

```
function getCheese(callback){
    setTimeout(()=>{
        const cheese="🧀";
        console.log("here is cheese",cheese);
        callback(cheese)
    },2000)
}
function makedough(cheese,callback){
    setTimeout(() => {
        const dough=cheese+"🥡";
console.log("here is dough",dough);
callback(dough);
    }, 2000);
}
function bakepizza(dough,callback){
    setTimeout(() => {
        const pizza=dough+"🍕";
console.log("here is my pizza ready😍",pizza);
callback(pizza);
    }, 2000);
}
getCheese((cheese)=>{
```



```
    makedough(cheese,(dough)=>{
        bakepizza(dough,(pizza)=>{
            console.log("got pizza😋",pizza);
        })
    })
})
```



```
here is cheese 🧀                                   script.js:124
here is dough 🧀🥛                                  script.js:131
here is my pizza ready😍 🧀🥛🍕                     script.js:138
got pizza😋 🧀🥛🍕                                  script.js:145
>
```

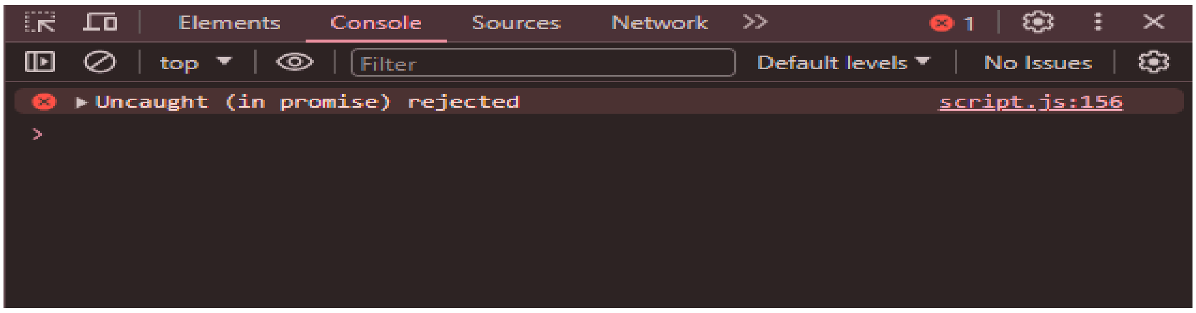
Promises

Promise is object in javascript , which has 3 states pending, resolve , reject

To create a promise object , we use the Promise() constructor.

```
let promise = new Promise(function(resolve,reject){


});
```

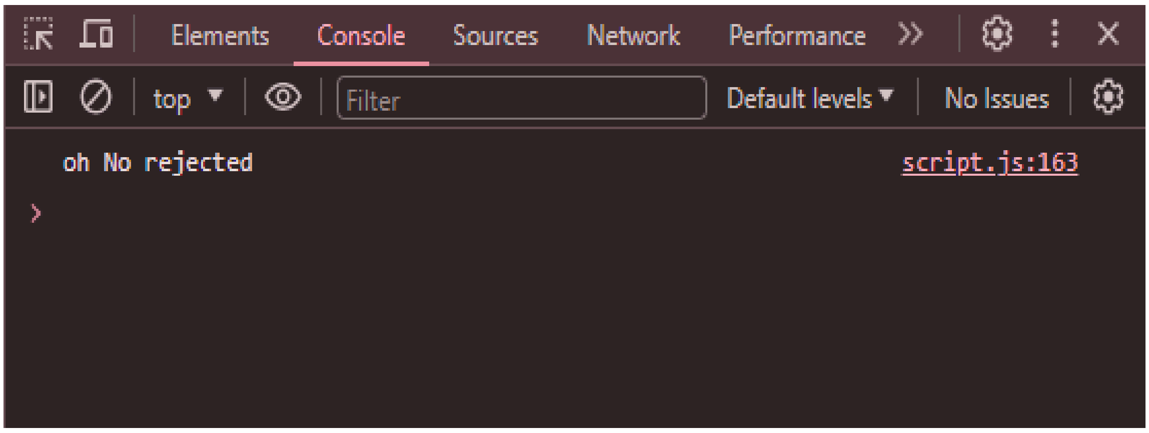
```
let promise = new Promise(function(resolve,reject){
    const state=false;
    if(state){
        resolve("success")
    }else{
        reject("rejected")
    }
});
```



```
▶ Uncaught (in promise) rejected                    script.js:156
>
```

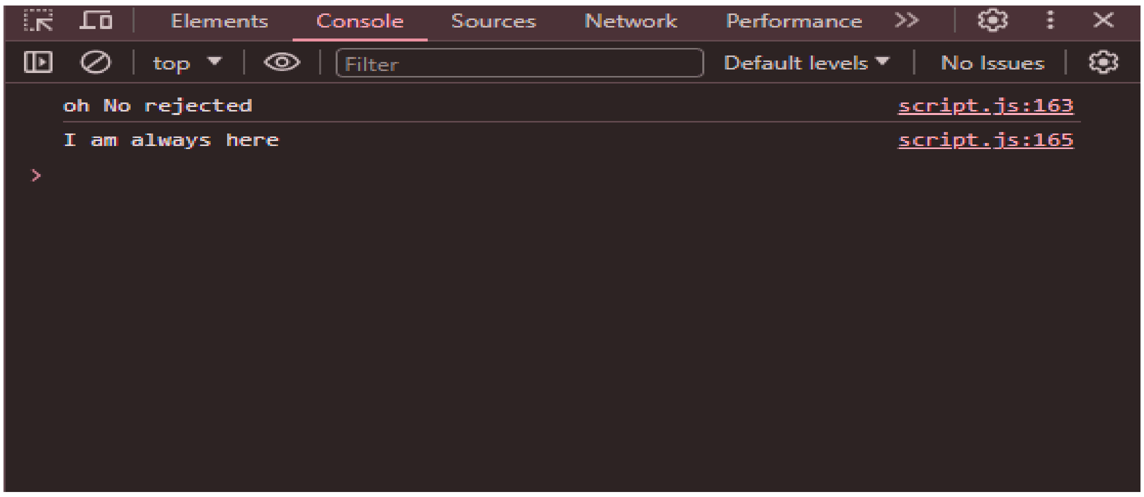
```
let demopromise = new Promise(function(resolve,reject){
    const state=false;
    if(state){
        resolve("success")
    }else{
        reject("rejected")
    }
});
```

```
demopromise.then((data)=>{
    console.log("wohoo great",data);
}).catch((data)=>{
    console.log("oh No",data);
});
```

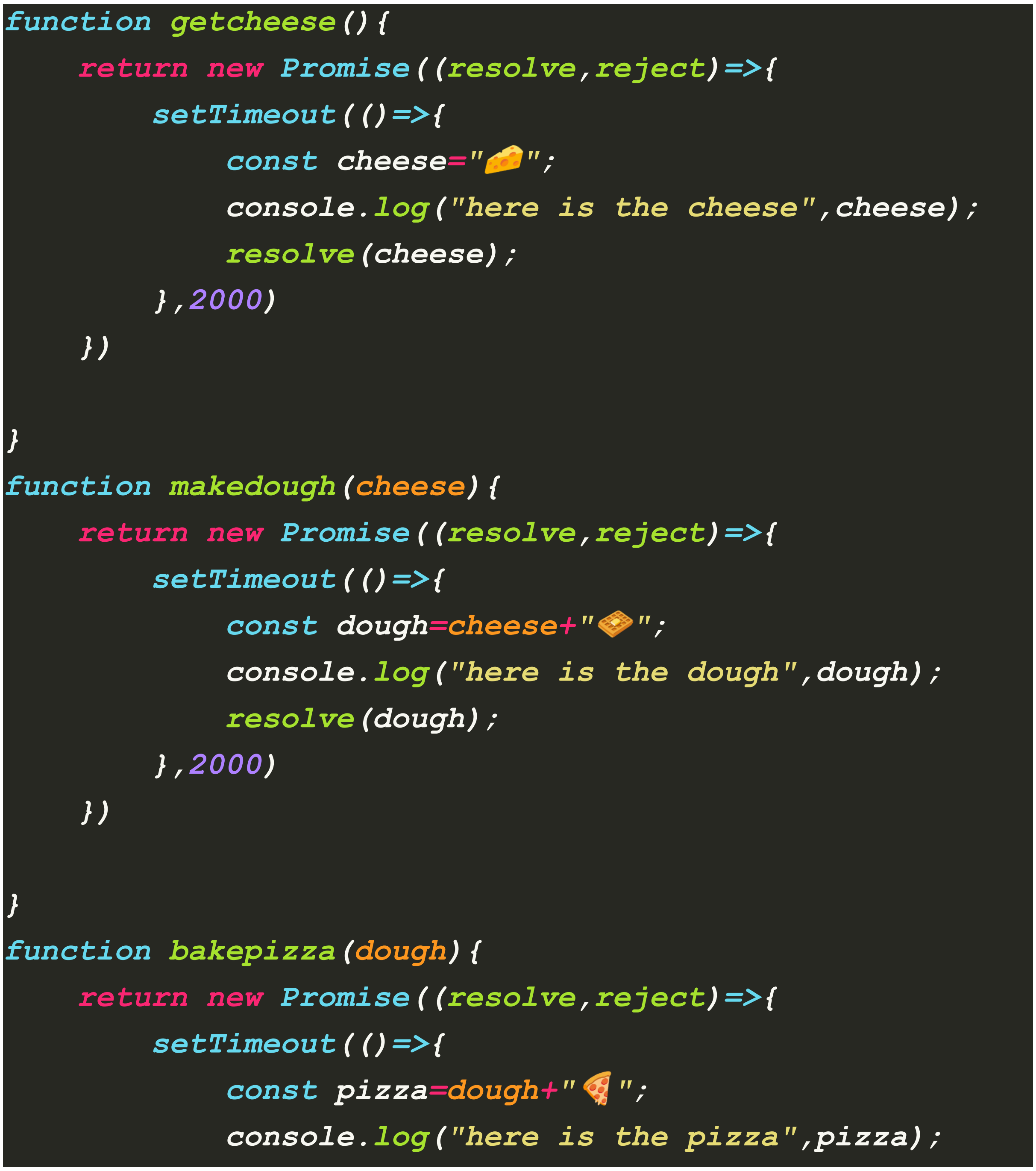


```
oh No rejected                                        script.js:163
>
```

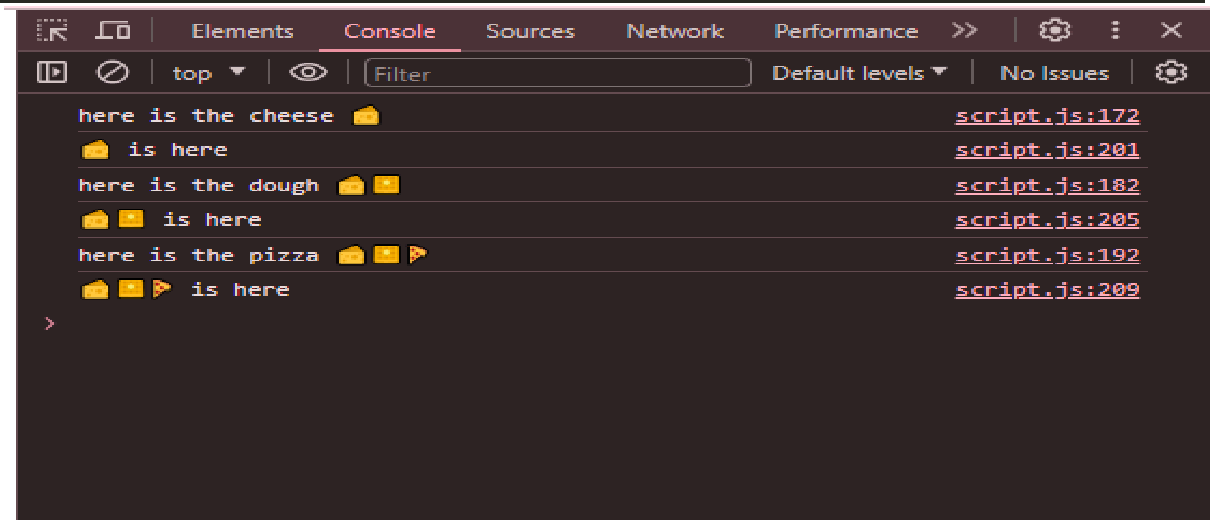
```
let demopromise = new Promise(function(resolve,reject){
    const state=false;
    if(state){
        resolve("success")
    }else{
        reject("rejected")
    }
});

demopromise.then((data)=>{
    console.log("wohoo great",data);
}).catch((data)=>{
    console.log("oh No",data);
}).finally(()=>{
    console.log("I am always here")
})
```



```
oh No rejected                                   script.js:163
I am always here                                 script.js:165
>
```



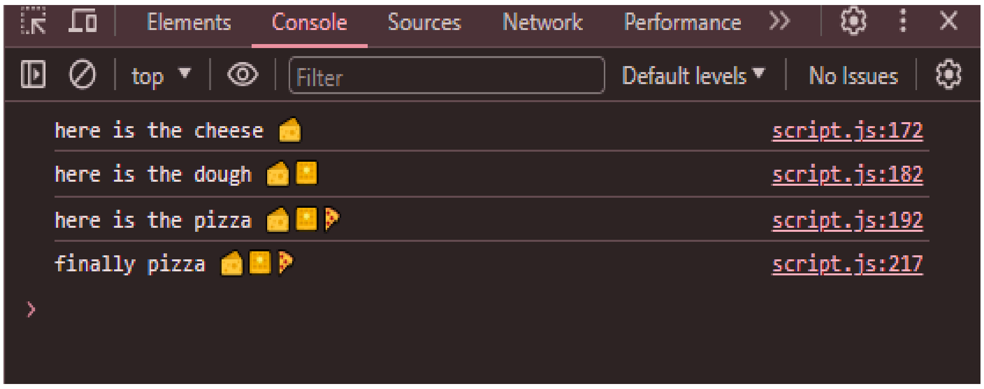
```
function getcheese(){
    return new Promise((resolve,reject)=>{
        setTimeout(()=>{
            const cheese="🧀";
            console.log("here is the cheese",cheese);
            resolve(cheese);
        },2000)
    })

}
function makedough(cheese){
    return new Promise((resolve,reject)=>{
        setTimeout(()=>{
            const dough=cheese+"🧇";
            console.log("here is the dough",dough);
            resolve(dough);
        },2000)
    })

}
function bakepizza(dough){
    return new Promise((resolve,reject)=>{
        setTimeout(()=>{
            const pizza=dough+"🍕";
            console.log("here is the pizza",pizza);
            resolve(pizza);
        },2000)
    })

}

getcheese()
.then((cheese)=>{
    console.log(cheese,"is here");
    return makedough(cheese)
})
.then((dough)=>{
    console.log(dough,"is here")
    return bakepizza(dough);
})
.then((pizza)=>{
    console.log(pizza,"is here")
}).catch((data)=>{
    console.log("Error")
});
```



```
here is the cheese 🧀                script.js:172
🧀 is here                           script.js:201
here is the dough 🧀🍞               script.js:182
🧀🍞 is here                         script.js:205
here is the pizza 🧀🍞🍕             script.js:192
🧀🍞🍕 is here                       script.js:209
>
```

Async await

```
async function orderPizza(){
    const cheese=await getcheese();
    const dough=await makedough(cheese);
    const pizza=await bakepizza(dough);
    console.log("finally pizza",pizza);
}
orderPizza();
```



```
here is the cheese 🧀                script.js:172
here is the dough 🧀🟨               script.js:182
here is the pizza 🧀🟨🍕             script.js:192
finally pizza 🧀🟨🍕                 script.js:217
```

```
async function orderPizza(){
    try{
        const cheese=await getcheese();
        const dough=await makedough(cheese);
        const pizza=await bakepizza(dough);
        console.log("finally pizza",pizza);
    }catch(error){
        console.log("error")
    }

}
```

DOM

Document Object Model

- **4 pillers of DOM**
    - Selection of an element
    - Changing HTML
    - Changing CSS
    - Event Listener

1.Selection of an element

```
document.querySelector("h4")

document.querySelector("#id")

document.querySelector(".class")
```

```
<script>
```

```
        const h1Element =
document.querySelector('h1');

console.log(h1Element); // Output: Hello, World!

h1Element.innerHTML="Shra";

h1Element.style.color="red"

h1Element.style.backgroundColor="black"

h1Element.addEventListener("click",function(){

    console.log("clicked..")

    h1Element.innerHTML="Shraddha";

h1Element.style.color="black"

h1Element.style.backgroundColor="yellow"

})

    </script>
```