

JAVASCRIPT :

SYNC :

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 asynchronous programming sometimes imp instructions get blocked due to some previous instructions, which causes a delay in the UI. Asynchronous code execution allows next instructions immediately and does not block the flow.

CALLBACKS :

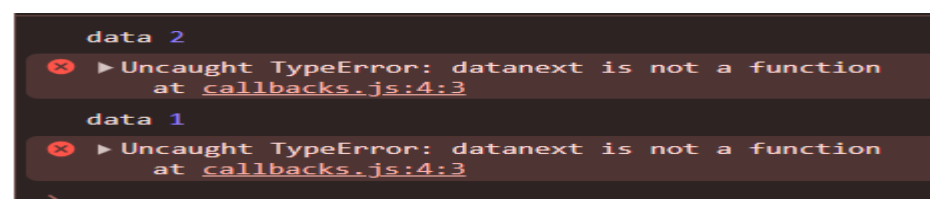
A callback is a function passed as an argument to another function.

CALLBACKHELL :

Nested callbacks stacked below one another forming a pyramid structure (pyramid dom).

This style of programming becomes difficult to understand & manage.

```
1  function getData(data, datanext){
2      setTimeout(()=>{
3          console.log("data", data)
4          datanext();
5      }, 2000)
6  }
7  getData(1, getData(2))
```



```
data 2
❌ ▶ Uncaught TypeError: datanext is not a function
    at callbacks.js:4:3

data 1
❌ ▶ Uncaught TypeError: datanext is not a function
    at callbacks.js:4:3
>
```

```

1  function getData(data, datanext){
2      setTimeout(()=>{
3          console.log("data", data)
4          datanext();
5      }, 2000)
6  }
7  getData(1, ()=>{
8      getData(2)|
9  })

```

```

data 1
data 2
✖ ▶ Uncaught TypeError: datanext is not a function
   at callbacks.js:4:3

```

PROMISE :

Promise is for “eventual” completion of task. It is an object in JS.

It is a solution to callback hell.

Function with two handlers

Let promise=new Promise((resolve,reject)=>{.....})

Resolve and reject are call backs provided by js.

```

12  let promise=new Promise((resolve,reject)=>{
13      console.log("i am student");|
14  })

```

```

i am student
> promise
< ▼ Promise {<pending>} i
  ▸ [[Prototype]]: Promise
    ▸ catch: f catch()
    ▸ constructor: f Promise()
    ▸ finally: f finally()
    ▸ then: f then()
    Symbol(Symbol.toStringTag): "Promise"
  ▸ [[Prototype]]: Object
  [[PromiseState]]: "pending"
  [[PromiseResult]]: undefined
>

```

There three states 1.pending 2.fullfilled 3.reject

```

12 let promise=new Promise((resolve,reject)=>{
13     console.log("i am student");
14     resolve("success");
15 })

```

```

i am student
> promise
< ▼ Promise {<fulfilled>: 'success'} i
  ▸ [[Prototype]]: Promise
    [[PromiseState]]: "fulfilled"
    [[PromiseResult]]: "success"
>

```

```

12 let promise=new Promise((resolve,reject)=>{
13     console.log("i am student");
14     reject("failed");
15 })

```

```
i am student
✖ ▶ Uncaught (in promise) failed
> promise
< ▼ Promise {<rejected>: 'failed'} i
  ▶ [[Prototype]]: Promise
    [[PromiseState]]: "rejected"
    [[PromiseResult]]: "failed"
  >
```

Promise.then((res)=>{...})

Promise.catch((err)=>{...})

```
17  const getPromise=()=>{
18      return new Promise((resolve,reject)=>{
19          console.log("i am student");
20          //resolve("success");
21          reject("error found")
22      })
23  }
24
25  let promise=getPromise()
26  promise.then(()=>{
27      console.log("promise fulfilled")
28  })
29
30  promise.catch(()=>{
31      console.log("error")
32  })
```

```
i am student
error
✖ ▶ Uncaught (in promise) error found
```

```
35 function asyncfunc(){
36     return new Promise((resolve,reject)=>{
37         setTimeout(()=>{
38             console.log("data failed")
39             resolve("success");
40         },2000)
41     })
42 }
43 let p1=asyncfunc();
44 p1.then((res)=>{
45     console.log("ok :-",res);
46 })
```

data failed

ok :- success

>

```
35 function asyncfunc1(){
36     return new Promise((resolve,reject)=>{
37         setTimeout(()=>{
38             console.log("data1 fetching")
39             resolve("success1");
40         },2000)
41     })
42 }
43 function asyncfunc2(){
44     return new Promise((resolve,reject)=>{
45         setTimeout(()=>{
46             console.log("data2 fetching")
47             resolve("success2");
48         },2000)
49     })
50 }
51 let p1=asyncfunc1();
52 p1.then((res)=>{
53     console.log("ok :-",res);
54     let p2=asyncfunc2();
55     p2.then((res)=>{console.log("ok :-",res);});
56 })
```

data1 fetching

ok :- success1

data2 fetching

ok :- success2

ASYNC-AWAIT :

async function always return a promise.

Sync function myFun(){.....}

Await pauses the execution of its surrounding async function until the promise is settled.

```
60 function api(){
61     return new Promise((resolve,reject)=>{
62         setTimeout(()=>{
63             console.log("api fetching....");
64             resolve(200);
65         },2000)
66     })
67 }
68
69 async function getdata(){
70     await api();
71     await api();
72 }
73
```

```
> getdata()
< ▼ Promise {<pending>} i
  ► [[Prototype]]: Promise
    [[PromiseState]]: "pending"
    [[PromiseResult]]: undefined
  ② api fetching....
>
```

IIFE :- immediately invoked function expression

IIFE is a function that is called immediately as soon as it is defined.