

4promise.html

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5     <meta charset="UTF-8">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <title>Document</title>
8 </head>
9
10 <body>
11
12     <script>
13         //technical suneja
14         //async js programming
15         //callback promise async and awaits
16
17         out ke sath //explanation this program jiski timing kar rahte hai wo hi pahle chalata hai sirf set time
18         // const datas=[
19         //     {
20         //         name:"Ajay", Profession:"software developer"
21         //     },
22         //     {
23         //         name:"anuj", Profession:"software developer"
24         //     }
25         // ]
26         // function getDatas(){
27         //     setTimeout(()=>{
28         //         let output=" "
29         //         datas.forEach((data,index)=>{
30         //             output+= `<li>${data.name}</li>`
31         //         })
32         //         document.body.innerHTML=output;;;;;;;;;;
33         //     },3000)
34         // }
35
36         // const createData=(newdata)=>{
37         //     setTimeout(()=>{
38         //         datas.push(newdata)
39         //     },2000)
40         // }
41
42         // getDatas()
43         // createData({name:"shubh",Profession:"web doveloper"})
44
45
46         //explanation this program jiski timing kar rahte hai wo hi pahle chalata hai sirf set time out
47         ke sath
48
49
50
51
52         //callback
53         // const datas=[
```

```

54 // {
55 //     name:"Ajay", Profession:"software developer"
56 // },
57 // {
58 //     name:"anuj", Profession:"software developer"
59 // }
60 // ]
61
62 // function getData(){
63 //     setTimeout(()=>{
64 //         output="";
65 //         datas.forEach((data,index)=>{
66 //             output+=`<li>${data.name}</li>`
67 //         })
68 //         document.body.innerHTML=output;
69 //     },1000)
70 // }
71
72 // function createData(newdata,callback){
73 //     setTimeout(()=>{
74 //         datas.push(newdata)
75 //         callback()
76 //     },3000)
77 // }
78 // // getData()
79 // createData({name:"shubh",Profession:"web developer"},getData)
80
81
82
83
84 //promise
85 // const datas=[
86 //     {
87 //         name:"Ajay", Profession:"software developer"
88 //     },
89 //     {
90 //         name:"anuj", Profession:"software developer"
91 //     }
92 // ]
93
94 // function getData(){
95 //     setTimeout(()=>{
96 //         output="";
97 //         datas.forEach((data,index)=>{
98 //             output+=`<li>${data.name}</li>`
99 //         })
100 //         document.body.innerHTML=output;
101 //     },1000)
102 // }
103
104
105 // function createData(newdata){
106 //     return new Promise((resolve,reject)=>{
107 //         setInterval(()=>{
108 //             datas.push(newdata);
109 //             let error=true;
110 //             if(!error){
111 //                 resolve()

```

```

112         //     }else{
113         //         reject("kuch sahi nahi hai....")
114         //     }
115         // },2000)
116         //     })
117         // }
118
119
120         // createData({name:"shubh",Profession:"web
doveveloper"}).then(getData).catch(err=>console.log(err))
121
122
123
124
125
126
127         //async and await
128
129
130
131         //master async sherians coding school
132
133         // sync matlab ek ke bad dusra hoga jab tak ek command complete na ho, dusra nahi hoga
134
135         //task a -5
136         //task b -2
137         //task c -15
138         //task d -1
139         //task e -4
140         // total time 27 sec
141         // console.log("hey1")
142         // console.log("hey2")
143         // console.log("hey3")
144
145
146         // async matlab saare kaam ek sath shuru kardo jiska answer pahle aajaye uska jawaab dedena
147
148
149         // kaise pata chalata hai ki hum sync code likh rahe ya async?
150
151         // async ki pahchan
152         // setInterval
153         // setTimeout
154         // Promise
155         // fetch
156         // axios
157         // XMLHttpRequest
158
159
160
161
162
163         // async js hai kya?
164         // kai bar aapko final code depended hota hai kisi aur ke server par, is case main hame pata
nahi hota ki ans uske server se kab laut kar aayega to hum kya nahi kr sakte is writing sync code, isse
nipatane ke liye hum log async code likh deta taaki blocking na hoo and jab bhi answer aaye humara
answer ke respect mai chalne wala code chal jaye
165
166         // async code ka main motive hota hai ki un case main jisme hume pata nahi code ka answer ke

```

respect main koi particular code chala dena

```
167
168 // example set time out
169 // callback function humesha async code mein answer ane par chalta hai
170 // console.log("hey1")
171 // setTimeout(function() {
172 // console.log("hey2")
173 // }, 2000)
174
175
176
177 // js is not asynchronous
178 // async ka matlab do kam ek sath karna
179
180 // jo bhi main stack par hota hai and jo bhi side stack par hota hai wo behind the scenes
prossing kar sakte hai aur jab uski processing complete ho use main stack mai la kar chalaya ja sakta
hai
181
182 // main stack execution stack
183
184 // event loop important for interview
185
186
187 // console.log("hey1")
188 // console.log("hey2")
189 // setTimeout(function() {
190
191 // console.log("hey3")
192 // }, 0)
193 // console.log("hey4")
194
195
196 // single threading and multi threading
197 // js single threading hai
198
199 // ye request bhejete hai
200 // callback
201 // fetch
202 // axios
203 // setTimeout
204 // setInterval
205
206 // jab complete ho toh iska ans yaha hai
207 // promise
208 // then catch
209 // try and catch
210 // async await
211
212
213
214 // callback => function
215 // callback hamesha ek function hota ye sirf tab chalta hai jab async code ja complete hojataa
hai
216
217 // example
218 // fetch('www.indiawalesir.com').then()
219 // var ans= new Promise((res, rej) => {
220 // if(true) {
221 // return res()
```

```

222     //     }
223     //     else{
224     //         return rej()
225     //     }
226 // })
227 // ans.then(function(){
228 //     console.log("resolve ho gya")
229 // }).catch(function(){
230 //     console.log("reject ho gay")
231 // })
232
233
234
235 // example
236 // user will ask for a number between 0 se 9 and if the number is below 5 resolve if not
reject
237
238 // var ans=new Promise((res,rej)=>{
239 //     var n=Math.floor(Math.random()*10);
240 //     console.log(n)
241 //     if(n<5){
242 //         return res()
243 //     }else{
244 //         return rej()
245 //     }
246 // })
247
248
249
250
251 // ans.then(function(){
252 //     console.log("below")
253 // }).catch(function(){
254 //     console.log("abobe")
255 // })
256
257
258 // what is promise
259 // a promise is an object that returns a value which you hope to receive in the future but not
now
260
261 // why we need promise
262 // javascript always synchronous and single threaded language. it means javascript never waits
for code or function result when they take some time js direct excute next code
263
264
265 // example 1 problem
266 // let data=1;
267 // console.log(data);
268 // data=2;
269 // setTimeout(()=>{
270 //     console.log("timer"+data) //output 3
271 // },2000)
272
273 // data=3;
274 // console.log("last"+data)
275
276
277 // promise example

```

```

278 // let data=new Promise((resolve,reject)=>{
279 //   setTimeout(()=>{
280 //     if(true){
281
282 //       resolve("done has been executed")
283 //     }else{
284
285 //       reject("code has been rejected")
286 //     }
287 //   },2000)
288 // })
289 // data.then((item)=>{
290 //   console.log(item)
291 // }).catch((err)=>{
292 //   console.log(err)
293 // })
294
295 // fetch ke andar promise wala code apne aap likha hota hai
296 //
297 // let data=fetch("http://127.0.0.1:5500/data.json");
298 // data.then((item)=>{
299 //   return item.json() //json ko resolve karne ke liye
300 // }).then((result)=>{
301 //   console.log("2nd out",result)
302 // })
303
304
305
306 // promise chaining
307
308 // let data=new Promise((res,rej)=>{
309 //   if(true){
310 //     res(10);
311 //   }else{
312 //     rej("reject ho gya")
313 //   }
314 // })
315 // data.then((item)=>{
316 //   console.log(item);
317 //   return item*10;
318 // }).then((item)=>{
319 //   console.log(item)
320 // }).catch((err)=>{
321 //   console.log(err)
322 // })
323
324
325
326
327
328 // finally keyword
329 // finnal block hamesha chalega chahe reject ho ya resolve ho
330
331 // let data=new Promise((resolve,reject)=>{
332 //   setTimeout(()=>{
333 //     resolve(10);
334 //   },2000)
335 // })

```

```

336 // data.finally((item)=>{
337 //     console.log("this is finnal block")
338 // }).then((item)=>{
339 //     console.log("this is resolve")
340 // })
341
342
343
344
345
346 // throw new error chalne par hamese catch block mai hi jayeha
347
348 // let data=new Promise((resolve,reject)=>{
349 //     setTimeout(()=>{
350 //         resolve("done");
351 //     },2000)
352
353 // })
354
355 // data.then((item)=>{
356 //     throw new Error("data issue")
357 // console.log("then block",item);
358 // }).catch((err)=>{
359 //     console.log("catch erroe",err)
360 // })
361
362
363
364 // promise.all
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391 // anuj bhaiya
392 // in Javascript a promise is a good way to handle asynchronous operation it is used to fined
out if the asynchronous operation is successfully completed or not.

```

```

393 // a promise may have three state
394 // pending    fulfilled    reject
395
396
397 // creating a promise
398 // to create a promise we use the Promise() constructor
399
400 // the promise constructor takes a function as an argument .
401 // the function also accepts two function resolve() and reject() if the promise return
successfully the resolve function is called and if an error occurs the reject function is called
402
403
404 // using a promise
405
406 // then() method
407 // the then method is used with the callbacks when the promise is succrssfully fullfiled or
resolve
408
409 // the catch() method
410 // the catch method is used with the callback when the promise is rejected or if an error
occurs.
411
412 // the finally() method
413 // the finally method gets executed when the promise is either resolved successfully or
rejected
414
415
416
417 // const ticket=new Promise(function (resolve,reject){
418 //     const isBoarded=true;
419 //     if(isBoarded){
420 //         resolve("your are not in the flight")
421 //     }else{
422 //         reject("your flight has been cancelled")
423 //     }
424 // })
425
426 // ticket.then((data)=>{
427 //     console.log("wohoo",data)
428 // }).catch((data)=>{
429 //     console.log("o no",data)
430 // }).finally(()=>{
431 //     console.log(" i will always be executed")
432 // })
433
434
435
436
437 // example 2
438 // function getCheese() {
439 //     return new Promise((reslove, reject) => {
440 //         setTimeout(() => {
441 //             const cheese = "🧀";
442 //             reslove(cheese);
443 //         }, 2000);
444 //     })
445 // }
446 // function makeDough(cheese) {
447 //     return new Promise((reslove, reject) => {

```



```

448 //         setTimeout(() => {
449
450 //             const dough = cheese + "🍞"
451 //             reslove(dough);
452 //         }, 2000);
453 //     })
454 // }
455 // function bakePizza(dough) {
456 //     return new Promise((reslove, reject) => {
457 //         setTimeout(() => {
458
459 //             const pizza = dough + "🍕"
460 //             reslove(pizza);
461 //         }, 2000);
462 //     })
463 // }
464
465
466
467
468 // getCheese()
469 //     .then((cheese) => {
470 //         console.log("here is your cheese", cheese)
471 //         return makeDough(cheese)
472 //     }).then((dough) => {
473 //         console.log("here is your dough", dough);
474 //         return bakePizza(dough)
475 //     }).then((pizza) => {
476 //         console.log("here is your pizza", pizza);
477 //     }).catch((data) => {
478 //         console.log("some thing error")
479 //     })
480
481
482 // example 3 using async await
483 // function getCheese() {
484 //     return new Promise((reslove, reject) => {
485 //         setTimeout(() => {
486 //             const cheese = "🧀";
487 //             reslove(cheese);
488 //         }, 2000);
489 //     })
490 // }
491 // function makeDough(cheese) {
492 //     return new Promise((reslove, reject) => {
493 //         setTimeout(() => {
494
495 //             const dough = cheese + "🍞"
496 //             reslove(dough);
497 //         }, 2000);
498 //     })
499 // }
500 // function bakePizza(dough) {
501 //     return new Promise((reslove, reject) => {
502 //         setTimeout(() => {
503
504 //             const pizza = dough + "🍕"
505 //             reslove(pizza);

```

```
506 //           }, 2000);
507 //           })
508 //       }
509
510 // async function orderPizza(){
511 //     try{
512 //         const cheese=await getCheese();
513 //         console.log("here is your cheese",cheese);
514 //         const dough=await makeDough(cheese);
515 //         console.log("here is your dough",dough);
516 //         const pizza=await bakePizza(dough);
517 //         console.log("here is your pizza",pizza);
518 //     }catch(err){
519 //         console.log("error happen",err);
520 //     }
521 //     console.log("ended")
522
523 // }
524
525 // orderPizza()
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
```

564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621

622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679

680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713 </script>
714 </body>
715
716 </html>