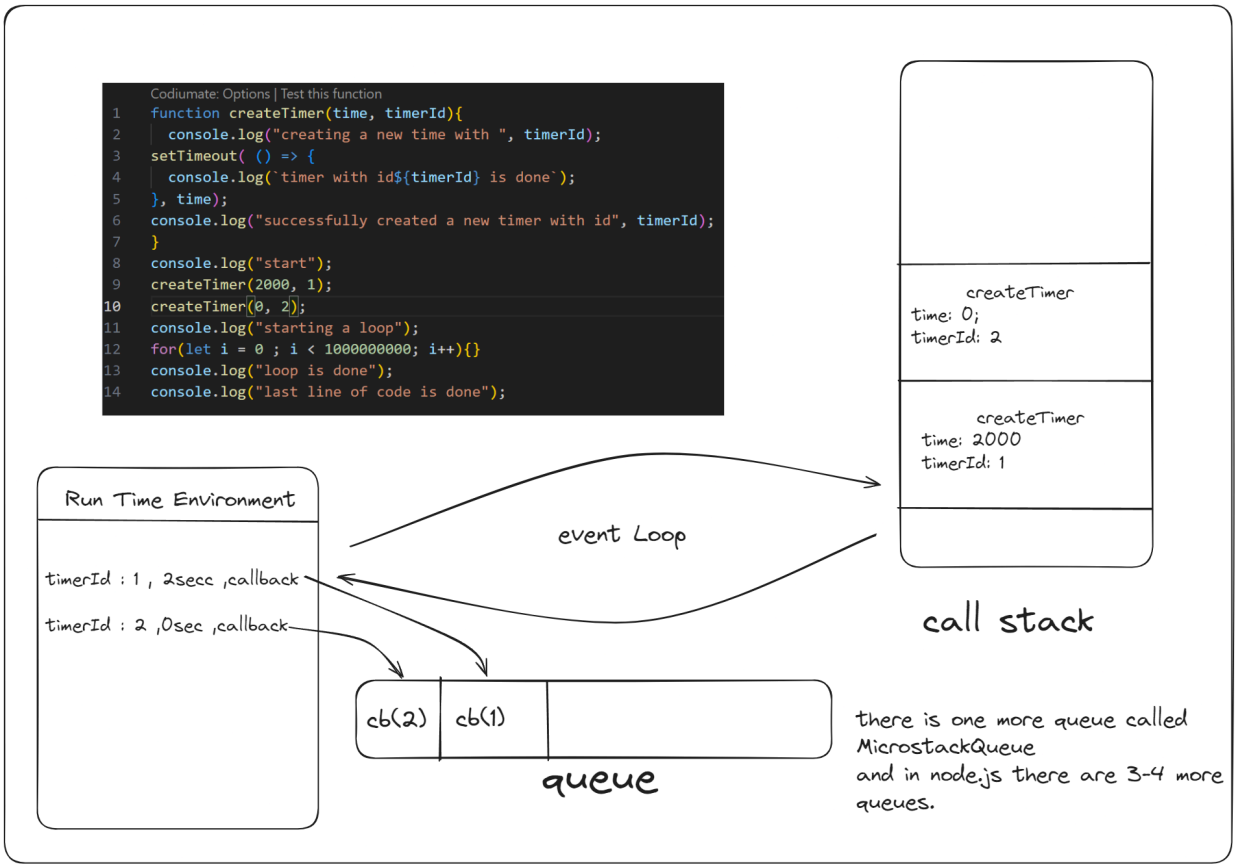
```
Codiumate: Options | Test this function
function createTimer(time, timerId){
  console.log("creating a new time with ", timerId);
setTimeout( () => {
  console.log(`timer with id${timerId} is done`);
}, time);
console.log("successfully created a new timer with id", timerId);
}
console.log("start");
createTimer(2000, 1);
createTimer(0, 2);
console.log("starting a loop");
for(let i = 0 ; i < 1000000000; i++){}
console.log("loop is done");
console.log("last line of code is done");
```




```
start
creating a new time with  1
successfully created a new timer with id 1
creating a new time with  2
successfully created a new timer with id 2
starting a loop
loop is done
last line of code is done
timer with id2 is done
timer with id1 is done
```

#How is our callback able to access local variable of a function taht does'nt exist beecause after execution of main thread the global lscope become empty .

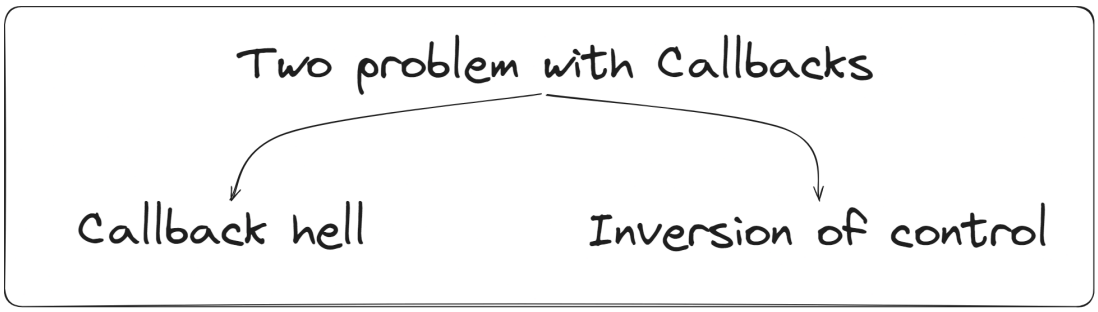
```
> function fun(c, d){
    let m = 10;
    function gun(){
      console.log("addition of m & c is", m + c);
    }
    return gun;
  }

  const g = fun(8, 5);
undefined
> g();
addition of m & c is 18                                    VM48:4
undefined
> console.dir(g);
                                                           VM110:1
  ƒ gun() i
    arguments: null
    caller: null
    length: 0
    name: "gun"
  ▶ prototype: {}
    [[FunctionLocation]]: VM48:3
  ▶ [[Prototype]]: ƒ ()
  ▼ [[Scopes]]: Scopes[3]
    ▶ 0: Closure (fun) {c: 8, m: 10}
    ▶ 1: Script {LoadTimeData: ƒ, g: ƒ}
    ▶ 2: Global {window: Window, self: Window, document: document, name: '', location: Location, …}
```

Closure

- It is a mechanism using which a function's inner function remembers all those variables which are defined in outer function scope even when the outer function execution is completed.

A closure is the combination of a function bundled together (enclosed) with references to its surrounding state (the lexical environment). In other words, a closure gives you access to an outer function's scope from an inner function. In JavaScript, closures are created every time a function is created, at function creation time. [MDN Reference]

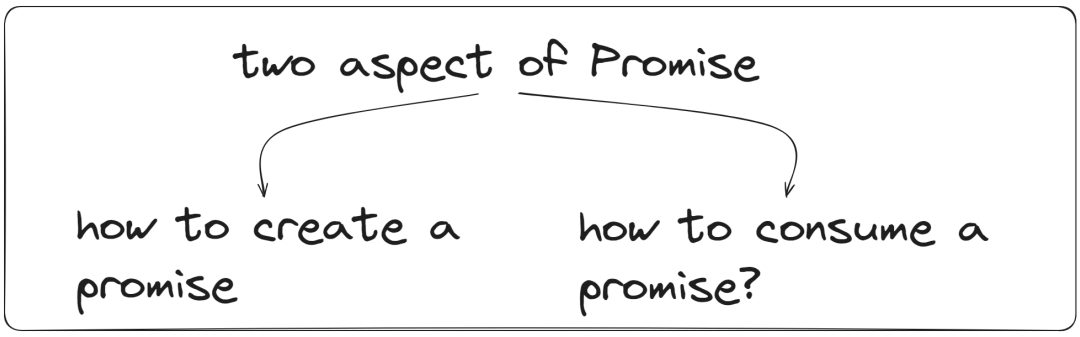


Promises : this is the official part of the JS documentation

- Can solve IOC(inversion of control) problem end to end & also solve a callback hell but generally promises don't solve callback problem
- It has its own problem (promise Hell).
- As it is the complete soln for IOC but not for callback hell still it is a good alternative to callbacks
- You can see promises in 70 percent of the production codebase.

Promises are readability enhancers

- They are a special object which can help us to control future related task.
- It's basically just a javascript object under the hood.



motivation problem statement
Implement a step of dummy functions which can mimic the behavior of following functions

- Download : should mimic download some context from url
- Write File : should mimic writing some content to a file.
- Upload : should mimic uploading to the file on any desire location a user wants

Now after you have implemented these functions try to use them in a scenario where we first download a file , then write it to a disk & then upload it to a server.

```
/* we don't want the download function hamper with
 main thread of the code and it should not block the
 main thread */
```

```
/**the process of downloading the data nad
 * hat to do with the downloaded data is
 *  independent of each other.
 * what to do after downloading can be decided
 * by whosoever is calling download function
```

```
 */
```

```
function download(URL, callback){
  console.log("downloading form URL", URL);
  setTimeout(() => {console.log("downloading is done");
    let  downloadedData = "someData" ;
    callback(downloadedData);
  }, 3000);
}

function writeFile(data, filename, callback){
  console.log("continuing", data ,"to file");
  setTimeout( () => {
    console.log("writing the file", filename,"is done");
    let status = "success";
    callback(status);
  }, 2000);
}

function Upload(fileName , url , callback){
  console.log("uploading file", fileName , "is done");
  setTimeout( () => {
    let status = "success";
    callback(status);
  }, 2000);
}

function process() {
  download("http",(data)=>{
    writeFile(data, "file.txt",(status)=>{
      Upload("file.txt", "http", (uploadstatus) => {
        console.log("all done");
      });
    });
  });
}
```