```
function printString(string) {
    setTimeout (
        () => {
            console.log(string)
        },
        Math.floor (Math.random () * 100) + 1
    )
}

function printAll() {
    printString("a");
    printString("b");
    printString("c");
}
printAll()
```

→ Independent with its own setTimout

X wait for function to finish

that's why we need Asynchronous!

Callbacks

```
function printString (string, callback){
  setTimeout(
    () => {
      console.log(string)
      callback( )
    },
    Math.floor(Math.random()*100)+1
  )
}

function printAll(){
  printString("A", () =>{
    printString( "B", () =>{
      printString("C", () => {} )

    })

  })
}
```

>< UGLY Code

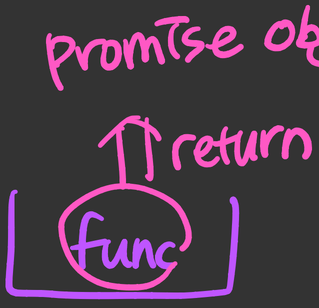
hard to read code.

callback Hell

printAll()

promises fix nesting probs.

```
function printString(string){
    return new Promise((resolve, reject)=>{
        setTimeout(
            () => {
                console.log(string)
                resolve()
            }
            Math.floor(Math.random * 100)+1
        )
    })
}
```



call
~~callback~~
↓
resolve
or
reject(error)

promise return will be promise

```
function printAll(){
    printString("A")           → will be promise
    .then(() => {                                  promiseobj .then
        return printString("B")
    })                          → promiseobj.then
    .then(() => {
        return printString("C")    → promiseobj
    })
}
printAll()
```

promise chain

X nested

x messy !

```
function printAll() {
    printString("A")
     .then(() => printString("B"))
     .then(() => printString("C"))
}

printAll()
```

Arrowfunc. ver

ERROR → .catch()

Await → Syntactic sugar for Promises

Async code ⇒ look more like sync, procedural code

easier to understand!

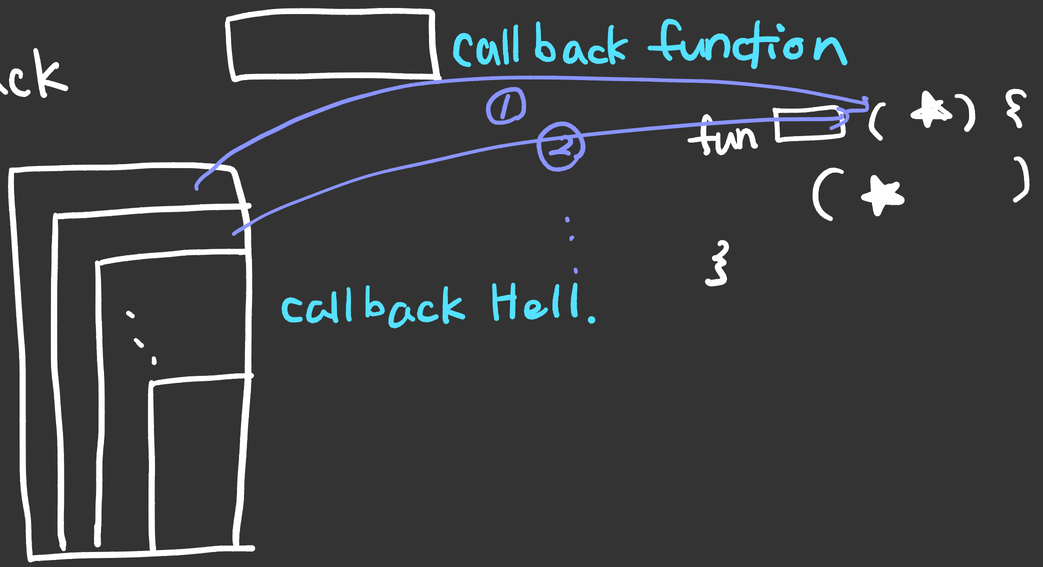
Promise methods can be used! | try catch keyword for wrapper func

```
async function printAll(){
    await printString("A")
    await printString("B")
    await printString("C")
}
```

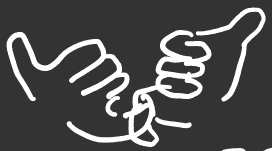
Much Better!

X use Await at Global Level!

call back



promise



ACCESS
.then()

ERROR
.catch()