27 Thursday

May 2021

Promise

```
const print = new Promise ((resolve, reject)=>{

    setTimeout(()=>{
    resolve (message)
     2000);

  })
 Console.log(Print);

   Print
      .then((response)=>{
        Console.log(response);
      ~~.catch()=~~
   .cath ((message) => {
          Console.log(message);
     })
```

28 Friday

then when a promise is successful, you can then use the resolved data.

Catch. when a promise fails, you catch the error and do something with the error information

Finally when a promise settles (fails or passes), you can finally do something.

May 2021

29 Saturday

Promise is in one of the 3 different states

· Pending - The initial state of promise.
· Fulfilled - The state of a promise representing a successful operation.
· Rejected - The state of promise representing a failed operation.

```
function Print(){

   ~~return~~ new Promise((res, rej)=>{
        setTimeout(()=>{
           message = `print successful'
            res(message)
           }, 2000);

    })
  }
 Print().then((res)=>{console.log(res);})

(new Promise(()=>{}) then(). catch()
```

30 Sunday

31 Monday

May 2021

```
// Fn handled by peintee

function Peint (data){
return new Peomise {(ees, eej) => {
     setTimeont () => {
     console . log (`peinting completed', data);
     message = `peint successful '
     ees (messag)
     }, 2000);
  })
 }
```

```
// one fn to staet peinting''
     function callPeintee (){
        const eesult = peint('sample')
       console . log (eesult);
       }
```

01 June Tuesday

```
     async function callpeintee(){
     const eesult = await peint ('sample')
     con sole . log (eesult);
     }.
```

The async/await syntax enables you to handle peomise without using .then() and .catch() method chaining, which also eemoves the need foe nested callbacks.

June	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
	22	23	24	25	26	27	28	29	30												

June 2021

02 Wednesday

Async makes a fn eetuen a peomise.
await makes a fn wait foe a peomise.

```
  function add (n1,n2){
   eetuen new peomise ((ees, eej) => {
         setTime out (()=> {
              let sum = n1 +n2
              ees (sum)
             }, 2000)
         })
       }
     Function mult (n1) {
     eetuen new Peomise ((ees, eej) => {
          SetTime out (() => {
             let peod = n1x100
             ees(peod)
            }, 20000)
          })
        }
```

03 Thursday

```
     function div(n1) {
  eetuen ~~P~~ new Peomise ((ees, eej) => {
       set Timeout (() =>{
          let value = n1/10
          ees ~~(ees)~~ (value)
          }, 2000)
        })
      }
```

June	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
	22	23	24	25	26	27	28	29	30												

04
Friday

```
asyn function calcutate(){
    const sum = await add(2,3)
    console.log({sum});

    const prod = await mult(sum)
    console.log({prod});

    const result = await div(prod)
    console.log({prod});
}
calculate()
```

only Final O/P

05
Saturday

```
    asynch function calculate(){
        const result = await ~~add~~ add (2,3)
        const result = await mult (sum)
        const result = await div.(prod)
        console.log({result});
    }
```

while using .then

```
add(2,3).then((sum)=> {
    mult(sum).then(prod)=>{
        div(prod).then(result)=>{
            console.log(result)};
        }})}
```

June T W T F S S M T W T F S S M T W T F S S M
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
22 23 24 25 26 27 28 29 30

06
Sunday

Exception handling

```
1. try {
        } catch (err){
                }

    try { console.log('start');
        let a=20, b=30
            let sum = a+b
            console.log(sum);
            console.log('end');

        } catch (err) {
            console.log(err);
        }
    console.log('Hai');
```

07
Monday

```
2. finally {
        console.log('finally');
    }
```

To intensionally make error

```
    try {
        console.log('start');
        let a=20, b=30
```

June T W T F S S M T W T F S S M T W T F S S M
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
22 23 24 25 26 27 28 29 30

Throw

The throw ~~error~~ statement allows you to create a custom error. The throw statement throws (generates) an error. The technical term for this is: The throw statement throws an exception.
The throw statement throws a user-defined exception. The execution of the current function will stop (The statements after throw won't be executed), and the control will be passed to the first catch block in the call stack.

APS (Application programming Interface)
Used to inter connect different softwares.

Browser APS

document.getElementById()
↙
API

Dom - Browser API
console -
local storage
geolocation

Fetch

Location

```
navigator.geolocation.getCurrentPosition(~~showPos~~)
                                   (showPosition ~~[illegible]~~
function showPosition (p) {
    const latitude = p.coords.latitude
    const logitude = p.coords.logitude
    console.log (latitude), (logitude)
}
```

```
function getCountries() {
    fetch('url').then((res)=>{
  })
  .catch(err => {
       Console.log(err)
  })
}
```

Fetch is a promise

json.

```
function getCountries() {
    Fetch('url')
    .then ((res) => {
       return res.json()
    }
    then ((result)=>{
       Console.log(result);
    })
```

June 2021

12
Saturday

```
.catch (err =>{
    console.log (err)
})
}
```

3) Json

Javascript object Notation (JSON) is a standard text-based format for representing structured data based on javascript object syntax.
It is a flexible format for data exchange that enjoys wide support in modern programming languages and software systems.

13
Sunday

5

It allows developers to store variable data types as human-readable code.
store and communicate data objects.

June	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
	22	23	24	25	26	27	28	29	30												