whenever there is a big promise chain, the chain at the last which has catch handles any chains error

* what to do if we want to proceed further if any of the step fails?

A: Then catch should be placed below then handles that error and which will any then below catch will process further.

Developer responsibility to see where your catch fits

~~Promise APIs~~.

Async Await S02.Ep04.

1) What is aysnc?

Async is a keyword used before function. Async function always returns a promise

Eg: 1

```
async function getData() {
  return new Promise((resolve, reject) => {
    resolve("Promise resolved");
  });
}
```

Then consume as normal promise

```
const dataPromise = getData();
dataPromise.then((data) => console.log(data));
```

* What if we don't return a promise in async fn?

A: Eg2

```
async fn getData2() {
  return 3;
}
```

It itself wraps the returned data into a promise, Still it would be consumed as normal promise,

Keyline is

async always returns a promise

(2) Async and Await

Async and await are used to handle promises.

Why ?

How we used to handle promises before async/await?

```
const prom1 = new Promise ((res, rej) => {resolve("Prom1")})

async function hp promBefAsyncAwait( ) {
    return prom1;
}

prom1.then((data) => console.log(data));
```

After async/await

await is a keyword which is only used in front of promises and in functions starting with async

```
async function handlePromise1 ( ) {
    const val await prom1;
    Console.log(val);
}
```

(3) Difference between normal handling of Promises and async/await handling promises.

```
const promEx = new Promise ( (resolve, rejekt) => {
    setTimeout ( () => {
        resolve("promise after 10s");
    }, 10000)
})

function getData(){
    promEx.then( (data) => console.log(data),
        console.log('Namaste');
}
```

o/p for the above case is

Namaste
Promise after 10s,

So JS engine does not wait and comes after 10s as expected.

(b) But what if we use await

```
const promAs = new Promise ((resolve, reject) => {
    setTimeout (() => {
      resolve ("promise after 10s with async and await")
    }, 10000);

async function getData() {
    const val = await promAs
      console.log('Namaste');
    }   console.log(val));
```

We generally expect to be Namaste / undefined but await stops the program and all the lines below it executes after promise gets the value and it is assigned to val.

(c) What if await has 2 times in code

```
async function getData() {
  console.log('Hello world');
  const val = await P;
  console.log('Namaste JS');
  console.log(val);

  const val2 = await P;
  console.log('Namaste JS 2');
  console.log('val2');
}
```

o/p

Helloworld
after 10s
Namaste JS
val1
Namaste JS 2
val2

dive deep into multiple promises

```
Const p1 = new Promise ((resolve, reject) => {
  SetTimeout (() => {
    Resolve ('Promise1 resolved value');
  }, 10000);
});

Const p2 = new Promise ((resolve, reject) => {
  SetTimeout (() => {
    resolve ('Promise2 resolved value');
  }, 5000);
});
```

Code

```
async function handleMultiplePromises () {
  console.log("Hello world");
  Const val = await p1;
  console.log('Namaste JS');
  Console.log(val);

  Const val2 = await p2;
  Console.log("Namaste JS2");
  Console.log(val2);
}
handleMultiple promises
```

O/P — After 10s

Namaste JS

p1 resolved

Namaste JS2

p2 resolved

ans:

It looks like async function is trying to resolve all promises before logging statements.

But !!

what if p1 is resolved after 5s and p2 is resolved after 10s.?

O/P

After 5s

Namaste JS 1

P1 resolved

After less/after 5s

Namaste JS 2

P2 resolved

What happens behind the scenes

Call stack does not have handle promise waiting in it for resolve, but it suspends and it comes back into call stack as soon as promise resolved

Eg: P1 after 5s

P2 after 10s

→ Call stack

0s handle Promise → goes out

5s handle Promise → comes in → goes out due to P2

10s handle promises → comes in.

Real world examples of async/await

HW How fetch works?

fetch is a promise on resolving gives response object

response object has body which is a readable stream.

fetch ⇒ Response.json() ⇒ jsonValue

```
const data = await fetch(API_URL);
const jsonVal = await data.json();
```

Error Handling

Uses try, catch.

```
try {
  const data = await fetch(API_URL);
  const jsonValue = await data.json();
  console.log(jsonValue);
} catch(err) {
  console.log(err);
}
```

If API_URL is invalid

Program stopped executing and quickly jumps to catch block

One more way to handle

handlePromise().catch()

Async/Await vs Promise.then/catch

Async await is just syntactic sugar. The only effect is happening in code but it always deals with promises.

* Use async/await whenever possible