Rahul Aher

Page No. | Date

①

● Javascript - Day 12.

● ASYNCHRONOUS JAVASCRIPT :-

→ We don't have the data right away. Javascript is a single threaded language, it knows nothing of the outside world.

● PROMISES.

→ A promise is an object that may produce a singl value sometime in the future.

Either a resolved value

or a reason why it's not resolved / rejected.

• 3 states of a promise.

★ Fulfilled ★ Pending ★ rejected.

★ But, we already have callbacks, why promises?

→ Promises were introduced in ES6 and are a bit more powerful, let's see how?

★ CREATE a promise

```
const promise = new promise ((resolve, rejected)=>{

        if(someCondition ) {
            resolved ("work");
        } else {
            reject ("something went wrong");
        }

});
```

Rahul Aher

Page No. (2)

Date

How to run Promise?

② get the result

```
Promise.then ( result => console.log(result));
```

① once promise is resolved or rejected

③ use the result.

Output : worked (assuming somecondition is true).

* Chaining in Promise :-

```
Promise.then (result1=> result1+ " :) "). then (
          ~~result2~~ result2  => console.log (result2);
  );
```

```
>Worked. :)
```

Explaination :- The first .then() gave us the result and it got passed on the second .then().
This ~~give~~ in chaining in promises.

* What if an error occurs in any of .then()?

```
Promise
      .then (....)
      .then (....)
         :
      .then(....)
      .catch( () => console.log('error'););
```

You can catch the error using catch.

→ .catch will only catch error of .then() before it. If

Rahul Aher

Page No. ③
Date

you have .then() after .catch() it won't catch the error [Try adding throw Error in .then()].

→ Promises are great for asynchronous programming.
* We can store a promises in a variable.
* We can do, .then() on a promise which can get executed when the promise return.

* Combining Promises :-

```
const promise1 = new promise ((resolve, reject) => {
        setTimeOut (resolve, 500, 'Hi P1")
  }

const promise2 = new promise ((resolve, reject) => {
        setTimeOut (resolve, 1000, `Hi P2');
  }

const promises3 = new promise ((resolve, reject) => {
        setTimeOut (resolve, 5000, 'Hi P3');
  }
```

→ To combine all these promises, we can use promise.all.

```
Promise.all ([promise1, promise2, promise3]).
        then ((values => { console.log(values); }
        );
```

→ It takes array ~~of arguments~~ as an ~~array~~ argument.

```
> ["HiP1", "Hi P2", "Hi P3"];
```

Rahul Aher

⇒ Returns an array of resolved values.

→ This result returns after 5000 ms.

Scanned by CamScanner