Asynchronous JavaScript

@codeWithSimran

↓

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 single 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 :-

```
                      create              takes 2 parameters
                      new promise         either resolve/reject
const promise = new Promise((resolve, reject)=>{
        if(someCondition) resolve("worked")
        else  reject("something went wrong")
    }
})
```

How to run the promise?

② get the result ↑ @codeWithSimran

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

① once promise is resolved or rejected

↓ ③ use the result

Output : worked (assuming some condition is true)

Chaining in Promises :-

→ sam as {return result1 + 😍}

```
promise.then(result1 => result1 + 😍)
 .then(result2 => console.log(result2))
```

```
> worked 😍
```

Explanation :- the first .then() gave us the result and it got passed on ~~th~~ to second .then()
This in chaining in promises.

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

@codeWithSimran

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

→ you can catch the error using .catch

.catch will only catch error of .then() ~~b~~ before it.
If you have .then() after .catch() it wont catch the error [Try adding throw Error in .then()]

Promises are great for asynchronous programming

* ★ we can~~t~~ store a promise in a variable
* ★ we can do .then() on a promise which can get executed when the promise returns

Combining promises :-

@codewithsimran

```
const promise1 = new Promise((resolve, reject) => {
        setTimeout (resolve, 500, 'Hi P1')
   }
const promise2 = new Promise((resolve, reject)=> {
        setTimeout (resolve, 1000, 'Hi P2')
   }
const Promise_3 = new Promise ((resolve, reject)=> {
        setTime out( 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 an array of promises as an argument

```
> ["Hi P1", "Hi P2", "Hi P3"]
```

★ Returns an array of resolved values

★ This result is returned after 5000 ms