Javascript

Javascript is a synchronous scripting language.

Functions

* Javascript functions are first class objects i.e., ability to be assigned to a variable, to be treated as a value, can have other functions within them, return functions to be called later

Callbacks

When a function accepts another function as an argument, this argument is known as a callback. A fundamental concept of functional programming.

E.g.,

let greeting = (name) => console.log(`Hello ${name}`);

const userInfo = (firstname, lastname, callback) =>{

    const name = `${firstname} ${lastname}!`

    console.log(name)

    callback(name)

}

userInfo('George', 'Chavady', greeting)

Promise

A Promise is used instead of a callback function. It has 2 callback functions as parameters – resolve and reject. Function.then() is used to get the resolve and ‘.catch(err)’ is used to handle the reject.

E.g.,

Function.then(res =>{

Console.log(res.<data>);

).catch(err =>{

Console.log(err.message);

})

E.g.,

const getTodo = () =>{

return new Promise((resolve, reject) =>{

    setTimeout(() =>{

        let error = false;

        if(!error){

            resolve({text: 'complete code example'})

        }

        else

            reject()

    }, 2000)

})

}

getTodo().

then(todo =>{

    console.log(todo.text)

})

.catch(()=>{

    console.log("error!")

})

**Note:** Template literals use a back tick/ back quote

Chaining of promises can be implemented using consecutive then statements.

const hasMeeting = true;

const meeting = new Promise((resolve, reject)=>{

    if(!hasMeeting){

        const meetingDetails = {

            name: "Scrum Meeting",

            team: "Production Support",

            location: "Skype",

            time: "14:30"

        }

        resolve(meetingDetails);

    }

    else{

        reject(new Error('Meeting already scheduled'));

    }

});

const addToCalendar = meetingDetails =>{

    const calendar = `${meetingDetails.name} is scheduled at ${meetingDetails.time} on ${meetingDetails.location} for team ${meetingDetails.team}`;

    return Promise.resolve(calendar);

}

meeting

    .then(addToCalendar)

    .then(res =>{

    console.log('Meeting Scheduled');

    console.log(res);

    // console.log('Name: ', res.name);

    // console.log('Time: ', res.time);

}).catch(err =>{

    console.log(err.message);

})

**Note:** the second then() statement takes in the result of the resolve from the first then().

Promise.all() is used to execute all the promises simultaneously.

Promise.race() is used to execute the first promise that is returned.

Promise.all([‘promise1’, ‘promise2’]).then(res => console.log(res));