**Error Handling**

**Errors not a mistake think it as features**

It has three properties

Ex : const myError = new Error(‘oops’);

Name of error ; myError.name//Error

Message : myError.message// oops

Stack trace : myError.stack// where the error happened

**Error** -> native error constructor functions

Should **throw** errors

Execution of the current process will be stopped

Throw anything :

***throw ‘string’ ; throw true ; throw new Error()***

function a(){

**const b = new Error('what ???');**

return b;

}

a(); **output**  Error: what ???

at a (<anonymous>:2:15) // function a on top of global execution context

at <**anonymous**>:5:1 //global execution context

If there is no catch for error in prg, then

browser catch : onerror() is executed (or)

node js catch : process.on(‘uncaughtException’)

**new Error**

**new SyntaxError**

**new ReferenceError**

**Try Catch:** 2 ways [ try{} throw catch{} finally] and catch() method

**Error handling for Asynchronous code**

**catch() method**

Asynchronous programming – **silent fail**

Node.js -> unhandledPromiseRejectionWarning -> if catch block is not present /you are not handling promises well

**Error handling for synchronous code**

**try {} catch (error){}**

catch has an error parameter (it is an **object )**

function fail() {

**try** {

console.log('this works');

**throw new Error('oopsie'); // to throw our error**

**} catch(e) {**

console.log('error', e);

**} finally {**

console.log('still good');

return 'returning from fail';

}

console.log('never going to get here'); // not executed

}

fail();

Can console.log

e.name

e.message

e.status

**Promises**

Promise.resolve('asyncfail')

.then(response => {

console.log(response)

**throw new Error('#1 fail')**

})

.then(response => {

console.log(response)

})

**.catch(err => {**

**console.error('error', err.message)**

**})**

.then(response => {

console.log('hi am I still needed?', response)

return 'done' **// this response is undefined**

**because last promise returned nothing**

})

.catch(err => { **// will not be called**

console.error(err)

return 'failed' })

**Async await**

// async await tries to convert async code in sync code

(async function() { // self invoked function here

**try {**

**await Promise.reject('oopsie')**

**} catch (err) {**

**console.error(err)**

**}**

console.log('This is still good!')

})()

**Extending Errors:** Error constructor is an object that we can extend from

this.name = 'PermissionError'

this.message = message

this.favouriteSnack = 'grapes'

}}

class DatabaseError extends Error {

constructor(message) {

super(message)

this.name = 'DatabaseError'

this.message = message

}}

**throw new PermissionError('A permission error')**

You should not return an error from node from your system with stack trace and all , as hackers would see your system details

**class authenticationError extends Error {**

**constructor(message) {**

**super(message)**

**this.name = 'ValidationError'**

**this.message = message**

**}}**

class PermissionError extends Error {

constructor(message) {

super(message)