performance

↳ measure speed of code

↳ how to write efficient & performing code

↳ event ~~for~~ loop

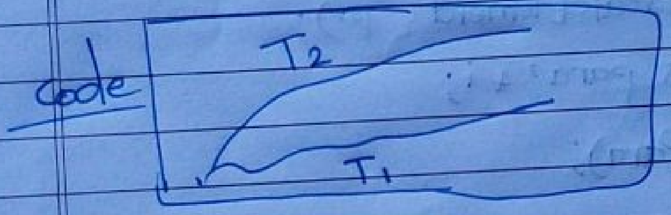
standard way to measure how long your code take to run.

performance.now();

bahot accurate 100%

reflow + repaint is very computationally intensive task

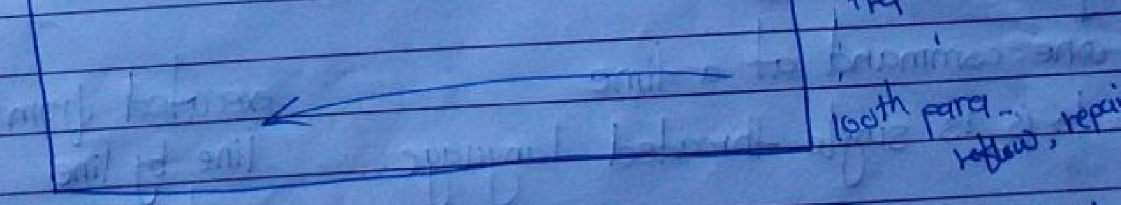
∴ try to keep less no. of reflow & repaints



∴ time taken : $(T_2 - T_2)$ ms

let say you are adding 100 para's to body..

Reflow → Repaint



for every para, there will be one reflow & one repaint..

reflow: Calculating the dimensions, position, size & all of the element.

repaint: drawing element on ur screen in pixels.

then what is best practice ?

→ Document Fragment
↓
light weight document object
↓
⊥ Reflow, ⊥ Repaint

if you add something then for that there will not be
Reflow ✗
Repaint ✗

e.g.
```
let fragment = document.CreateDocumentFragment();
for(let i=1; i<=100; i++)
{
  let newPara = document.createElement('p');
  newPara.textContent = 'This is para' + i;
  Fragment.appendChild(newPara);
}
document.body.appendChild(fragment);
```
↑
This will reduce no. of Reflow & Repairs

single threading
↓
one command at a time
↳ js is single threaded language..

executed from top to bottom line by line
↓
∴ synchronous language.

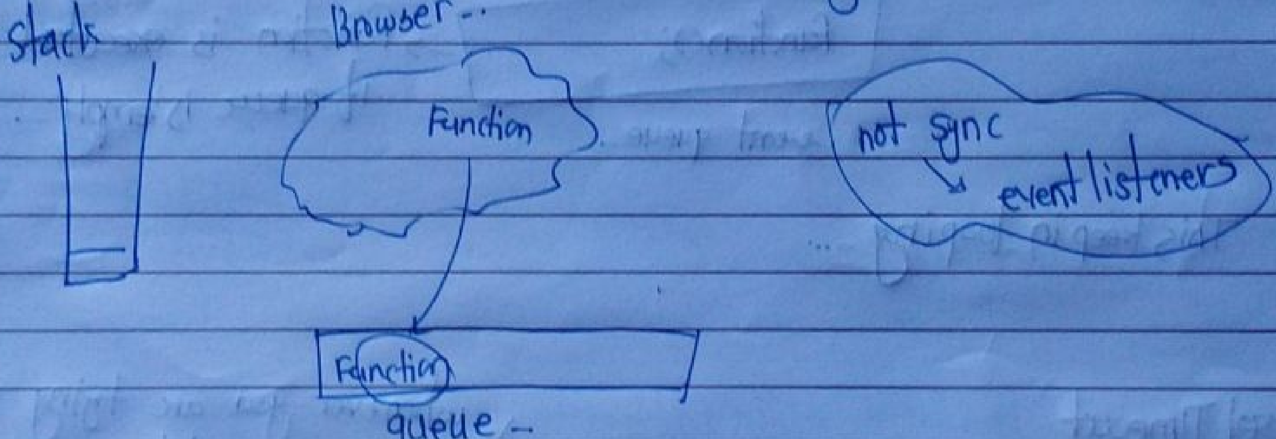
Page No.:
Date: / /

```
→ a();          function a();
                   b()
                   a()
```

Event loop

↳ call stack
↳ browser
↳ event queue

```
① console.log('Hi');  ①
② addeventlist('click', function(){
                     log('123')});  ③
③ console.log('Hello');  ②
```

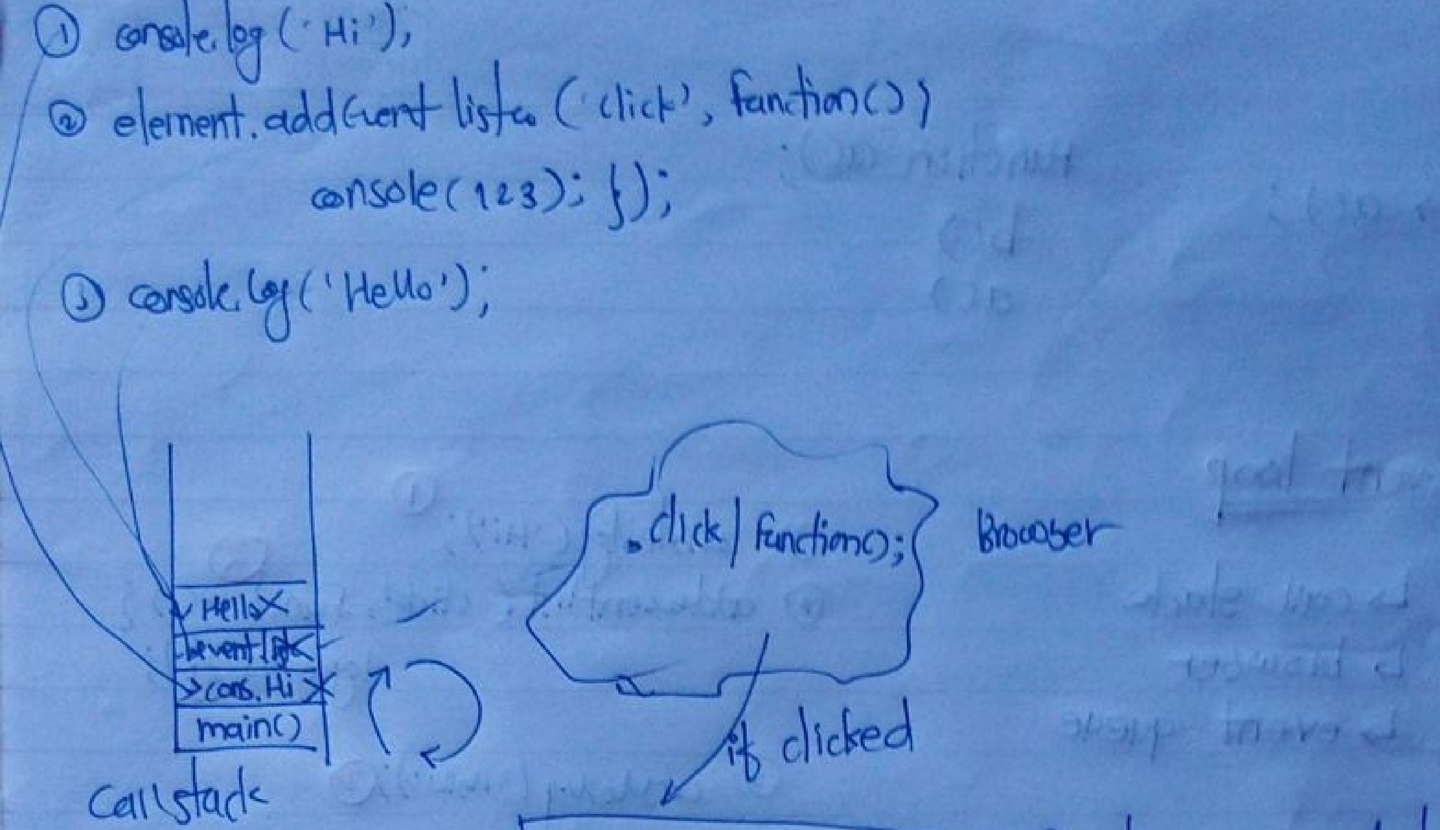


Async ---> eventlisteners etc.

① Async code → JS event loop

② any async code is handled by browser

① console.log('Hi');
② element.addEventlisten('click', function()}
　　　　console(123); });
③ console.log('Hello');



This keep in looping....

setTimeout

```
setTimeout( function(){
    console.log( .. );
}, 4000);
```

setTimeout(function(){ ← whenever you are trying to defer something until stack is clear..
　- - -
}, 0);
↓
passing 0 dosen't mean it will immediately run a function..
↓
this is async function
↓
will enter into $\boxed{\text{event loop}}$

async-await

special syntax used to work with promises

Lucky Page No.:
Date: / /

when 1 async code is running & you want another to wait for first code then you use async-await.

you can make any function async.

add keyword

e.g.
```
async function abcd() {
        console.log(...);
    }

console.log(abcd());

o/p : ▸ Promise
        Fulfilled
         value
```

```
eg. let mahara = new promise((resolve, reject) => {
        setTimeout(() => {
          resolve(---);
        }, 1000);
    };

    let hyd = new Promise(--
          ·==-
        };
```

API

Lucky Page No.:
Date : / /

Features of Async code
↳ clean & concise
↳ better error handling.

Promise

$$\begin{tikzcd} \text{Promise} \arrow[r] & \left.\begin{array}{l}\text{pending}\\ \text{fulfilled}\\ \text{rejected}\end{array}\right\} \text{3 statues} \end{tikzcd}$$

↳ when you want to keep running a thing parallely in the background in js..

```
let merapromise = new promise( function (resolve, reject){
                    console.log(---);
                     resolve(198);
                  )};
```

```
meraPromise;
▸ {<fulfuiled>, 198}
```

when want to reject..

```
setTimeout(function (resolve, reject) {
        consle.log( .. );
         reject(new Error('error aagi hai');
      },2000};
              ↓
          rejected..
```

Methods on promises

used after execution of promice.

- then → to get access of value returned by promise.
- catch → when error occurs..

```
meraPromise.then ((value) => { console.log(----));
meraPro.catch ( (error) => { console.log(-----);}
```