DOM + Modern JS - 3

#. Performance

- → Measure Speed of Code.
- → How to write efficient & performing code.
- → Event loop.

* performance.now() :-

In JavaScript, performance.now() method can be used to check the performance of your code. You can check the execution time of your code using this method.

Syntax :- `let t = performance.now();`

→ It does not take any parameter.
→ It returns value of time in milliseconds.

Example 1 :-

```
let t1 = performance.now();
for(let i=1 ; i<=10 ; i++)
{
    let newElement = document.createElement('p');
    newElement.textContent = 'This is Para ' +i ;
    document.body.appendChild (newElement);
}
let t2 = performance.now();
console.log ("This took " + (t2-t1) + " miliseconds ");
```

Output :- This took 0.4000000 miliseconds

// Example-2 Optimising Code

```
let t3 = performance.now();
let mydiv = document.createElement('div');
for(let i=1 ; i<=10 ; i++)
{
    let element = document.createElement('p');
    element.textContent = "This is Para "+i ;
    mydiv.appendChild(element);
}
document.body.appendChild(mydiv);
let t4 = performance.now();
console.log("This took " + (t4-t3) + "milliseconds");
```

Output:- This took 0.2000000 milliseconds

REFLOW :-

Reflow is the name of the web browsers process for re-calculating the positions and geometrics of elements in the document, for the purpose of re-rendering part or all of the document.

Reflowing is very expensive. Reflow occurs when you :-

→ insert, remove or update an element in DOM.
→ Move a DOM → resize the window etc.

REPAINT :-

Repaint is merely the changing of pixels on the monitor.
Repaint occurs when some changes which only its skin styles, such as color and visibility.
Repaint is faster as compared to Reflow.

* NOTE :- For Best practice approach, There should be less use of Reflow and Repaint.

Last 2 Codes have 10 Reflow and 10 Repaint, it can be any number for any time.

So, we use the concept of Document Fragment
↳ Only 1 Reflow & 1 Repaint.

Document Fragment :-

It is used as a lightweight version of document that stores a segment of a document structure comprised of nodes just like a standard document.

Example :-

```
let fragment = document.createDocumentFragment();
for(let i=1 ; i<=10; i++) {
    let newElement = document.createElement('p');
    newElement.textContent = 'This is Para ' +i;
    fragment.appendChild(newElement);
}
document.body.appenchild(fragment);
```

The Call Stack :-

* Single-Threading :-

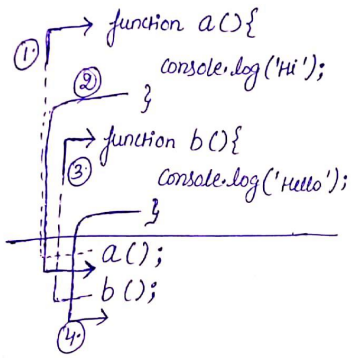
→ JavaScript is a single-threading language.

→ Single-Threading refers to the processing of one command at a time.

Observation :-

① "Run to Completion" Nature of Code.

② JS does not execute multiple lines / functions at the same time.



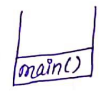
---

example of Call Stack

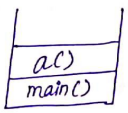
```
function a(){
    console.log('Hi');
    b();
}
function b(){
    console.log('Hello');
}
a();
```

(i) Empty Call Stack

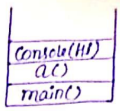
(ii) main() called



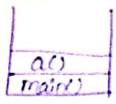
(iii) a() called.



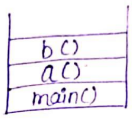
(iv) execute console.



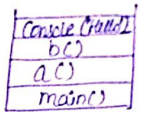
(v) After console Execution, Remove Console.



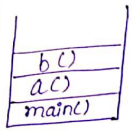
(vi) b() called



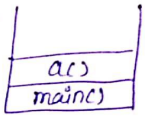
(vii) execute Console.



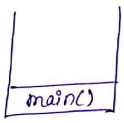
(viii) After execution Removed console



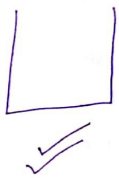
(ix) b() executed Completely



(x) a() executed Completely



(xi) Call Stack Empty After all execution Successfully.



Note:-

* Execution take place Top to Bottom. Therefore, execution is sequencewise or Synchronus in Nature.
* Called Synchronus Execution.

EVENT LOOP

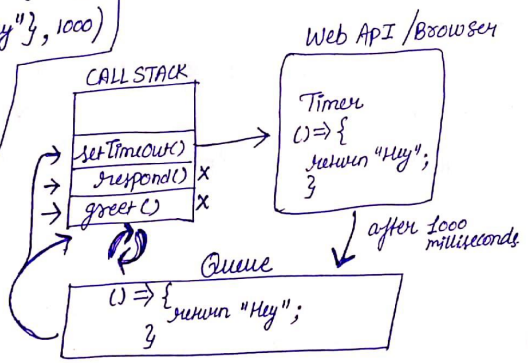
→ Event loop in JavaScript is a mechanism through which the 'calls waiting for execution' in the callback queue / job queue can be put on the call stack.

→ For any event from the callback queue / job queue to come to call stack, when call stack will be empty.

Example of Mechanism of Event Loop :-

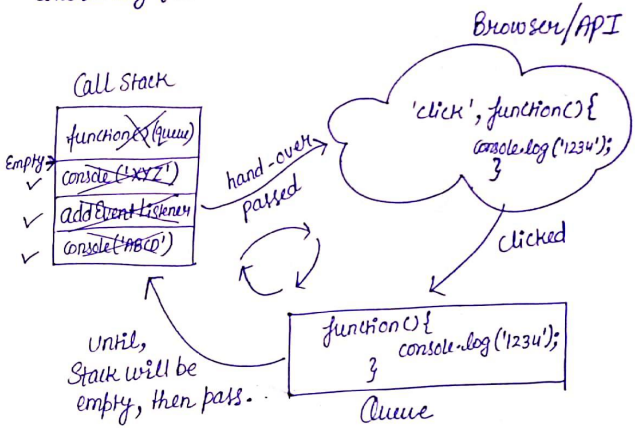
(1.)

```
* function greet(){
    console.log("Hello!");
  };
  function respond(){
    return setTimeOut(()=>{
      return "Hey"}, 1000)
  };
  greet();
  respond();
```



(2.)

```
console.log('ABCD')
element.addEventListener('click', function(){
                                    console.log('1234');
                                  }
                        );
console.log('XYZ');
```



Event Loop

Note:-

(1.) Any Async-code (Event Listener) is handled by Browser.

(2.) Async-code is executed through the Event Loop.

(3.) Whenever call stack empty, Async code passed to call stack from the queue. Then execute.

The SetTimeout() method :-

This method calls a function after a minimum number of milliseconds.

1 second = 1000 milliseconds.

Syntax :- `SetTimeout (function, milliseconds);`

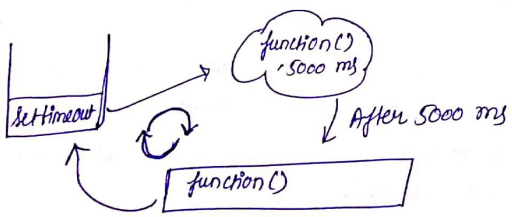
* Default value of millisecond = 0.
* Function has to wait atleast till given milliseconds. It is minimum wait time. It can be more.

example :-

```
SetTimeout (function () {
        console.log ('Hello Everyone');
}, 5000 );
```

Output :-

Hello Everyone → after 5 seconds of execution delay.



* Special Case :-

```
SetTimeout (function () {
}, 0);
```

↓

runs immediately ?? Concurrency ?

→ Answer :-

* SetTimeOut is an Async-Code, it goes to EventLoop.
* Then, SetTimeout wait in the queue for execution whenever Stacks empty.
* Therefore, It does NOT runs immediately at 0 milliseconds.
* Concurrency in Javascript is ability to execute Multiple tasks simultaneously. It is based on event loop.

Conclusion :- whenever you defer (टालना) something, we can use SetTimeout function.