17 | 48th Day Wk 07
SATURDAY ▪ FEBRUARY

Mon		5	12	19	26
Tue		6	13	20	27
Wed		7	14	21	28
Thu	1	8	15	22	
Fri	2	9	16	23	
Sat	3	10	17	24	
Sun	4	11	18	25	

February 2018 | March 2018

47) Higher Order functions in JS.

HOF is any function that takes one or more functions as argument which it uses to operate on any set of data and/or returns a function as a result.

Eg:- Map, forEach, reduce, filter takes an array and function as argument, uses that function to transform the array and return new transformed data.

48) Diff. b/w forEach and map.

Both of these methods are used to iterate over elements of an array but forEach does not return anything whereas map returns a modified array.

18 | 49th Day
SUNDAY

49)
```
console(1)
setTimeout(() => console(2), 0) → console(3)
new Promise((resolve) => {
    console(4)
    resolve()).then(() => console(5))})
console(6).
```

Output → 1 3 4 6 5 2 → why-?
All the normal consoles will be called first.
2018 Promise have higher priority than setTimeout even

ore functions
on any
function as a

an array
that function
new transformed

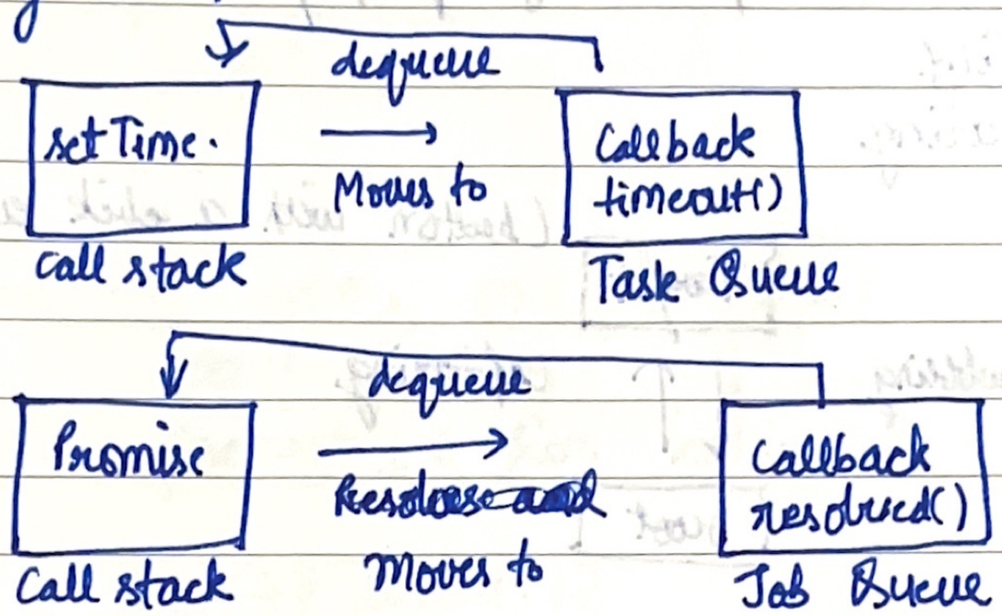
iterate over
does not return
a modified

console(3)

sole(5)})

alled first.
tTimeout even

if setTimeout is set to delay of 0.
The call stack is LIFO which stores execution contexts created during execution. Task queue and Job queue are FIFO, task queue holds callbacks from async operations and job queue stores callbacks from promises.



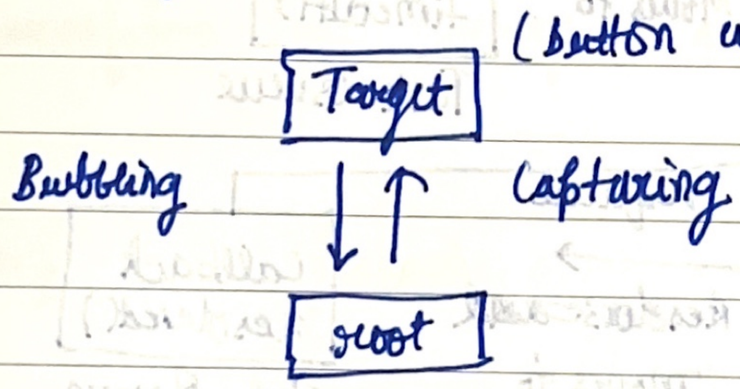
Now, event loop monitors callstack and once all created contexts are moved from callstack, it then looks into job and tasks queue. The event loop prioritizes jobs over tasks and deques the promise callback and push it back in call stack. Then call stack executes the resolved promise. Then it dequeues timeout() from task queue into callstack. At last, callstack executes timeout() callback.

50) Event propogation in Js.

Propogation refers to how events travel through DOM. The events passes to each node on DOM until it reaches the end or stopped forcibly.

There are two phases of propogation :-

1) Bubbling
2) Capturing



A target is the DOM node on which you trigger any event.
A root is the highest level parent of that object which is usually document. When we click on a button, a button with click event would be the event target.
Once we trigger the event, it propogates up the root and will trigger every single event handler which is associated with same type. When we triggered button click, it will trigger every click event along the way.
To stop this problem, we use event.stopPropogation().
This will prevent further propogation through DOM tree and only run event handler from which it was clicked.

February 2018

Mon		5	12	19	26
Tue		6	13	20	27
Wed		7	14	21	28
Thu	1	8	15	22	
Fri	2	9	16	23	
Sat	3	10	17	24	
Sun	4	11	18	25	

March 2018

	5	12	19	26	Mon
	6	13	20	27	Tue
	7	14	21	28	Wed
1	8	15	22	29	Thu
2	9	16	23	30	Fri
3	10	17	24	31	Sat
4	11	18	25		Sun

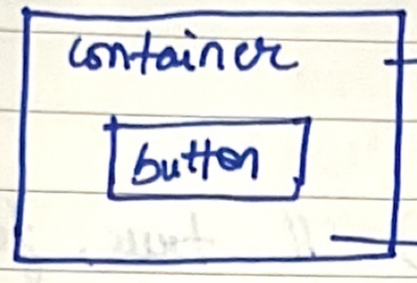
Wk 08 52nd Day

FEBRUARY ▪ WEDNESDAY 21

DOM. The
it reaches the

ont)

trigger any event.
t object which
a button,
event
e root and
which is
riggered button
along the
op Propogation().
through
handler



container.addEventListner("click", first());

button.addEventlistener("click", second());

```
function first (){
    log("1")
}

function second(){
    log("2")
}
```

Now, on clicking button, only 2 should be printed but due to event bubbling, 1 will also be printed. To prevent 1 & 2 from calling:-

```
f ~~first()~~ & second(){
    event.stopPropogation() -> log("1")
}
```

Now, here there is one thing. stopPropogation() prevents all click events on the parent node but not any other event handlers (e.g. when we have two clicks on button).

```
button.addEventlistener("click", third());
```

Now, 2 should be printed but it will print 1,2,3. stopPropogation() prevents 1 but still 2,3 will be printed. In this case, we use event.stopImmediatePropogation(): This will really stop parent click handlers and any other handlers.

2018

51)
```
var a = [1,2,3]
var b = a
var c = [...a]
console.log(a===b, a===c)  // true, false
```

52)
```
var a = [1,2,3,4,2]
a.filter((item, index) => a.indexOf(item) === index)

// [1,2,3,4]
```

Because for index = 4, item = 2, a.indexOf(2) returns first instance of 2 i.e. index = 1.

So,

$1 \neq 4$.

53)
```
f.a(){
  var func = [];
  for(var i=0; i<10; i++){
    func[i] = f.b(){
      return i
    }
  }
  return func.
}

const func = a();
func[10] => 10
func[5]  => 10
```

Because, var i makes global i and curry instance of i points to 10. To solve this, we can use let !

54)
```
console.log
```

55)
```
var x = 1
var y =
console.
```

(unary op
into nu
does same.

56) Polyfill i

Polyfill is a
to old brow
ES7 ... featu
polyfill.
firstly, we
```
f mapPol
  va
  for
  }
}
f callba
  ret
}
```

54) console.log(null == undefined) // true
(null === undefined) // false
(null == false) // false
(null == 0) // false
(undefined == 0 / false) // false

55) var x = 1
var y = "1"

console.log(++x, ++y) // 2 2

(unary operator, increment decrement converts string into number). Apart from this, * *, -, / also does same. only + coerces into string.

56) Polyfill in Js.

Polyfill is a piece of code which provide modern functionality to old browsers. Like old browsers may not support ES6, ES7 ... features. To help browsers support it, we use polyfill.

Firstly, we can directly create replica functions - like :-

```
function mapPoly(arr, callback){
    var new_arr = [];
    for(var i=0; i<arr.length; i++){
        new_arr.push(callback(arr[i]))
    }
}
function callback(item){
    return item * 2;
}
```

// Replicating map() functionality and passing a callback function on it. Now we can call :-

→ mapPoly(arr, callback);

2018

24 55th Day Wk 08
SATURDAY · FEBRUARY

Mon		5	12	19	26
Tue		6	13	20	27
Wed		7	14	21	28
Thu	1	8	15	22	
Fri	2	9	16	23	
Sat	3	10	17	24	
Sun	4	11	18	25	

February 2018

- We can also use a transpiler to this. A
- Babel transpiler is a free open source tool that converts ES6 code to backward-compatible versions of JS.

npm i @babel/cli @babel/core @babel/preset-env.

↓

This will also install babel-polyfill.

→ Now, we add es2015 to predefined presets.
→ Now, File create → .babelrc. Write code:-

{ "presets": ["@babel/env"] }.

- Create your JS file with normal ES6 code. To convert this in ES5, use command:-

npx babel <ES6 file> --out-file <new file-ES5>

- Include this new ES5 file along with babel-polyfill file from node module in index.html file.

25 56th Day SUNDAY
Run.

<script type="text/javascript" src=" ">

Note:- In earlier example, we saw that we can create replica of any function manually. What if, we want those manual functions to be called in same manner as ES6 methods are called. Then we assign it in Array.prototype

Array.prototype.map = function mapPolyfill(){...}

2018 Now, whenever we use our.map(), if any browser

March 2018

does
this
will

57) Cle

There
respecti
ClearTi
is a

does not have default map() in Array.Prototype then this polyfill map will be added and browsers will work in same manner as ES6.

57) clearTimout and clearInterval.

These both functions are used to stop execution of their respective set methods.

clearTimeout() stops execution of setTimeout(). SetTimeout() is a function which creates a delay of given time.

```
counter = 0;
timedCount();
function timed_Count(){
        counter++;
        timer_id = setTimeout(timed_Count, 1000);
}
// Timed_count function is called after delay of 1 sec
   which again calls timed_count and repeats. To break
this,
        clearTimeout(timer_id);
```

It will clear the timer set for setTimeout.

Similarly, clearInterval clears timer set for setInterval().

setInterval() calls some execution after every given seconds.

```
const id = setInterval(call(), 1000);
function call(){
        if(width = 50){ clearInterval(id); }
        width++;
}
```

2018

27 58th Day Wk 09

TUESDAY ▪ FEBRUARY

Mon		5	12	19	26
Tue		6	13	20	27
Wed		7	14	21	28
Thu	1	8	15	22	
Fri	2	9	16	23	
Sat	3	10	17	24	
Sun	4	11	18	25	

February 2018

	5	12	19	26	
	6	13	20	27	
	7	14	21	28	
1	8	15	22	29	
2	9	16	23	30	
3	10	17	24	31	
4	11	18	25		

March 2018

58) Parse method in JS. Date.

Date.parse() method parses the date and returns total milliseconds since Jan 1, 1970.

Date.parse('Jan 1, 1970, 00:00:00') → 0

Date.parse('04 Dec, 1995 00:12:00') → 818000253---

59) toDateString() → it returns the date part of Date() object in string.

60) Callback Hell.

Nesting of callbacks can lead to unreadable code called callback hell or pyramid of doom. This means, we call callback functions inside another callback.

```
f a(){
    do1(x,(do2)=>{
        do3(do2,(do4)=>{
            ...
        })
    })
}
```

pyramid of doom.

In this case, we will also have to deal with errors at each level instead of just at top level. Thus, we use promises for async programming in JS.

61)

9 → Primiti

10 → Size at th

11 → Alloca amou

12 All valu

1 (object, refere

2 heap i refere

3

62) va

4

5 va

6 3

7

63) TD A T is ina Variabl that of hoi

nd returns

.

SD) → 0

OD) → 818000253---

data part of
ing.

dable code called
This means,
ther callback.

sm.

lead with
at top level.
programming

61)

Stack	Heap
→ Primitive value and references	→ objects and functions
→ Size allotted is known at the compile time.	→ Size known at run time.
→ Allocates a fixed amount of memory	→ No limit per object

All values first point to the stack. If its non-primitive (object, array, functions), the stack contains a reference to object in the heap. The memory of heap is not ordered, thus we need to keep a reference to it in stack.

62)
```
var obj1 = {
    a:1
}
var obj2 = {
    a:1
}
obj == obj2   // false
obj1 === obj2  // false.
```

63) TDZ – Temporal dead zone.

A TDZ for any variable is the zone where the variable is inaccessible and returns reference error. TDZ for any variable starts with { and ends where we initialize that variable. for let and const but for var, because of hoisting it starts and ends at {.

64) Sort lexically in JS.

Using sort() method, it sorts the original array.

```
arr = [2, 3, 1, 5];
arr.sort()
console.log(arr)    // [1,2,3,5];

arr = ['banana', 'mango', 'apple'];
arr.sort()  // ['apple', 'banana', 'mango'];

var obj = [{ name: "a" },
           { name: "b" }]
```

Imp. obj.sort((a,b) => a.name.localeCompare(b.name))

65) What is event delegation?

Event delegation is a pattern based on event bubbling. It is an event-handling pattern that allows to handle events at higher level in DOM tree other than level where the event was fired. It helps create cleaner and shorter code. The idea is to "delegate" the handling of an event to a different element.

```
div.addEventListener("click", (event) => {
    if(event.target.name == "BUTTON"){
    }
})
```

02 61st Day Wk 09

FRIDAY ▪ MARCH

Mon		5	12	19	26
Tue		6	13	20	27
Wed		7	14	21	28
Thu	1	8	15	22	29
Fri	2	9	16	23	30
Sat	3	10	17	24	31
Sun	4	11	18	25	

March 2018

66) Flatten an array without flat().

Using concat:-

```
var arr = [1,[1,[1,2],2],2]
var a = [].concat(...arr)      [1,[1,[2,2],2]
var b = [].concat(...a)        [1,1,1,2,2,2]
```

or:-

```
arr.flat(2).
```

Using map:-

```
var f = arr.toString().split(',')
console.log(f.map(n => +n))
```

67) Prototype design pattern.

Prototypes allows us to hide complexity of making new instance. The concept is to copy existing object (acting as a prototype) into newly copied object which may change same properties only if required. These are required when object creation is time consuming and costly.

2018

2	9	16	23	Mon
3	10	17	24	Tue
4	11	18	25	Wed
5	12	19	26	Thu
6	13	20	27	Fri
7	14	21	28	Sat
8	15	22	29	Sun

1) Type of

2) Ways to

new obj

Object

var

var obj=

3) first class

first class

like a va

argument

and ca

any variable

first order

function a

as retur

const