1. Global Execution Context Lexical environment
   a) Creation phase

	TDZ
LE: {makeArmy:fn, outer:null} thisBinding:window}	army

   b) Execution phase

LE: {makeArmy:function, outer:null,army:[fn,fn]} thisBinding:window

2. **MakeArmy** Function Execution Context – Lexical Environment

   a. Creation

	TDZ
LE:{arguments:{length:0},outer:global}, thisBinding:window	Shooters i

   b. Execution

LE:{outer:global,shooters:[fn,fn], arguments:{length:0}, ~~i:0~~, ~~i:1~~, i:2}, thisBinding:window

   NB: since i is used after incrementation done, I putted the value lastly used.

3. While loop Execution Context
   a. Creation phase

	TDZ
LE:{outer:makeArmy,anonymous:fn}, thisBinding:window	Shooter

   NB: There are two iteration with the same lexical execution.

   b. Execution

   The first while loop execution context

LE:{ outer:makeArmy, arguments:{length:0},anonymous:fn,shooter: fn{alert(i)}}, thisBinding:window

   The second while lexical execution context

LE:{ outer:makeArmy,
arguments:{length:0},anonymous:fn,shooter:[fn{alert(i)}, fn{alert(i)}],
thisBinding:window

NB: i is incremented here.

4. Lexical execution for **army[0]**
    a. Creation phase

Closure scope:

i=2, outer:makeArmy

LE:{arguments:{length:0},outer:closurescope},
thisBinding:window

    b. Execution

Closure scope:

i=2, outer:makeArmy

LE:{arguments:{length:0},outer:closurescope}, alert(i)

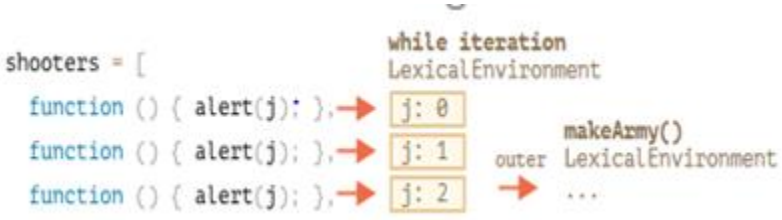
5. What will army[0] alert?
   Army[0] is a reference, however army[0]() is function call. Hence the call will print i=2.
6. Code fixed as below

```
function makeArmy() {
    let shooters = [];
    let i = 0;
    while (i < 2) {
        let j = i;
```

```
        let shooter = function() {
            console.log(j);
        };
        shooters.push(shooter);
        i++;
    }
    return shooters;
}
let army = makeArmy();
army[0]();
army[1]();
```

.

7. Changes in diagram after correction is all about having local variable **j** in addition.



Question 2:printNumber function that can print every number as scheduled.

```
function printNumbers(from,to){
let timerID= setInterval(()=>{
    if(from<=to){
        console.log(from);
        if(from==to){
            clearInterval(timerID);
        }
        from++;
    }
},3000)};
printNumbers(5,7);
```

Question 3: Explanation about setTimeout scheduled function.

```
let i=0;

setTimeout(()=>alert(i),100);

for(let j=0;j<100000000;j++){
```

```
  i++;

}
```

The scheduled function run after the the current code (i.e. the loop) finished. Because setTimeout simply queues the code to run once the current call stack is finished executing. Once done, the setTimeout will execute the alert which will be popout window with the value of i (i.e. 100000000).