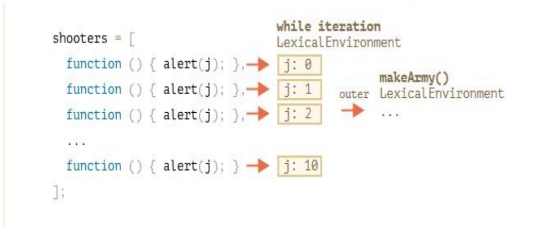
Homework

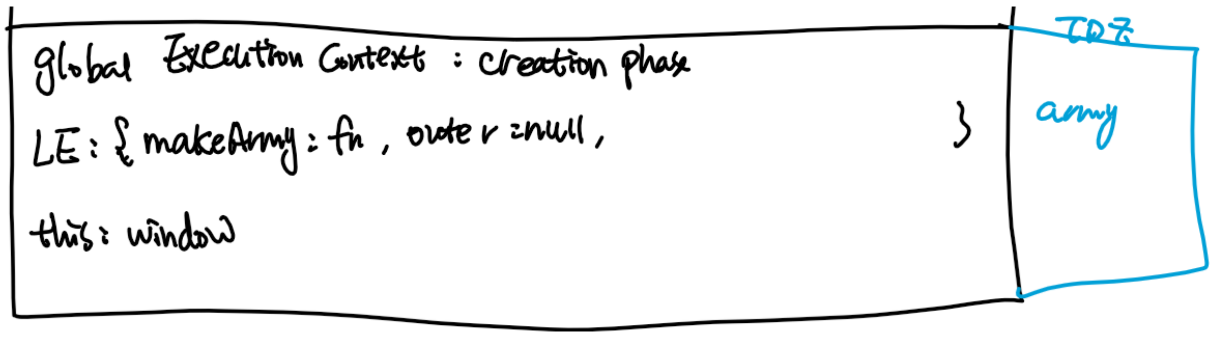
- Draw a lexical environment diagram for the right code and show:
  - global lexical environment (LE)
  - LE for makeArmy()
  - LE for LE of the while loop
  - LE for army[0]
  - What will army[0] alert?
  - Can you fix the code?
  - How will the diagram change?

```
function makeArmy() {
    let shooters = [];
    let i = 0;
    while (i < 2) {
        let shooter = function() {
            alert(i);
        };
        shooters.push(shooter);
        i++;
    }
    return shooters;
}
let army = makeArmy();
army[0];
```



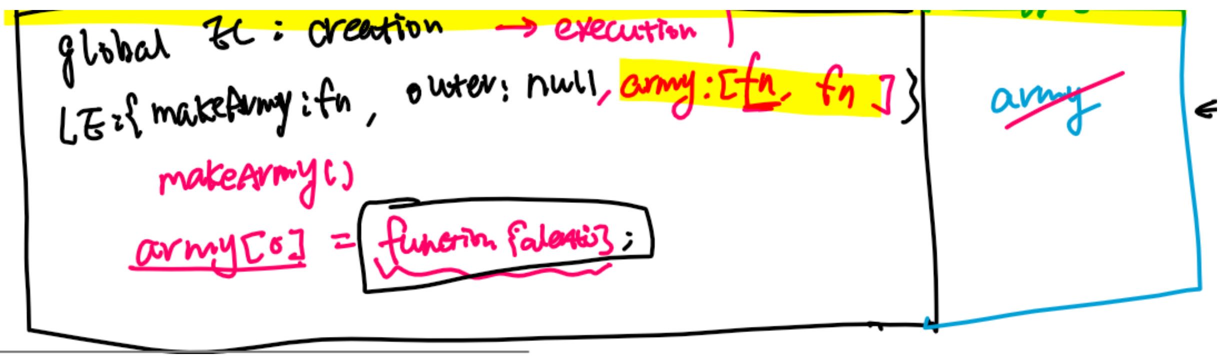
shooters = [
function () { alert(j); },
function () { alert(j); },
function () { alert(j); },
...
function () { alert(j); }
];
while iteration
LexicalEnvironment
j: 0
j: 1
j: 2
j: 10
outer
makeArmy()
LexicalEnvironment
...

- global lexical environment (LE)
  - creation phase



global Execution Context : creation phase
LE: { makeArmy: fn , outer: null, }
this: window
TDZ
army

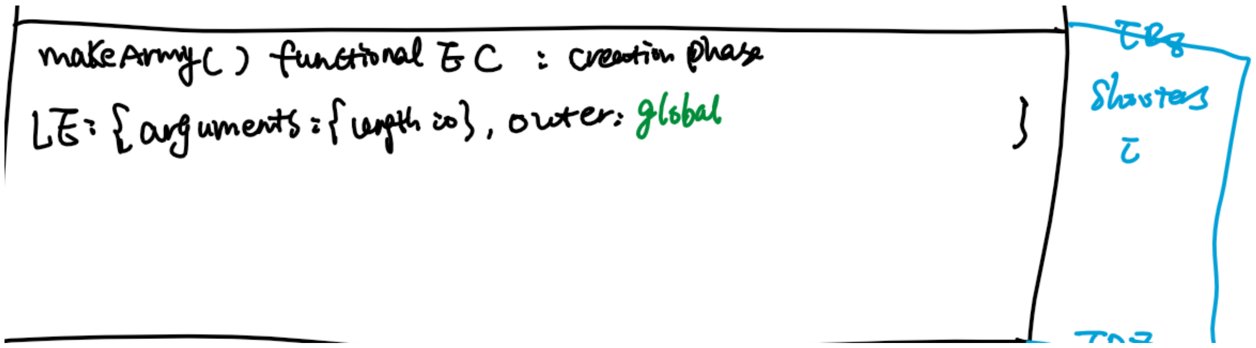
  - execution phase



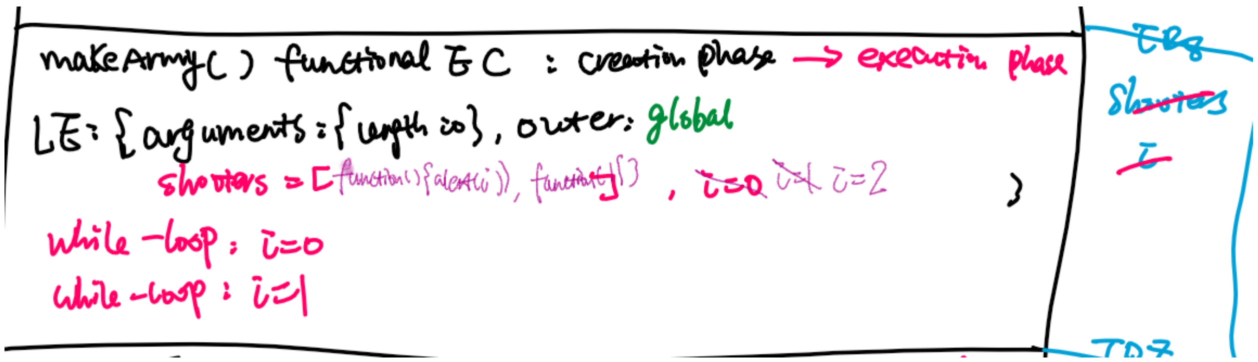
global EC : creation → execution
LE: { makeArmy: fn , outer: null, army: [fn, fn] }
makeArmy()
army[0] = function {alert(i)};
~~army~~

- LE for makeArmy()

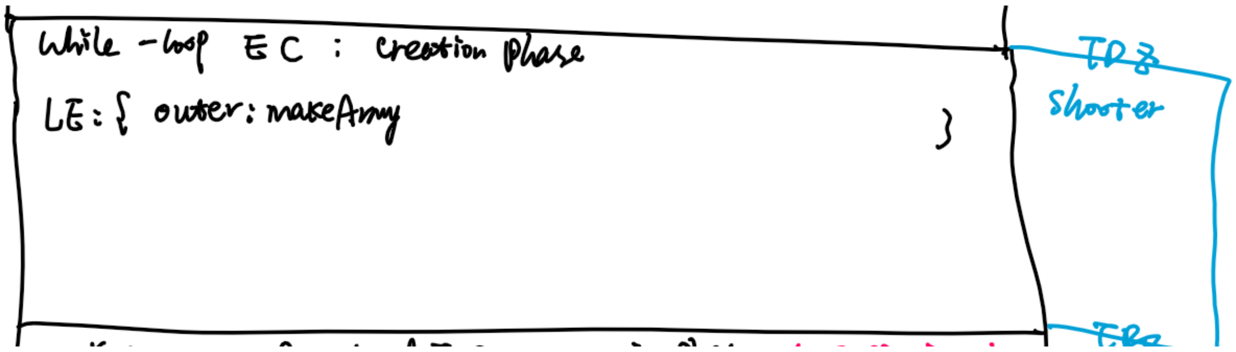
- creation phase


makeArmy( ) functional EC : creation phase
LE: {arguments: {length: 0}, outer: global }
ERs
Shooters
i
TDZ

- executon phase
  - After while-loop, changed the condition to while(i<2) to save time. The LE is being changed as below


makeArmy( ) functional EC : creation phase → execution phase
LE: {arguments: {length: 0}, outer: global
shooters = [function(){alert(i)}, function(){}] , ~~i=0~~ ~~i=1~~ i=2 }
while-loop: i=0
while-loop: i=1
ERs
~~Shooters~~
~~i~~
TDZ

- LE for LE of the while loop
  - Each iteration of while loop has own LE, only use i=0 as example here.
  - creation phase

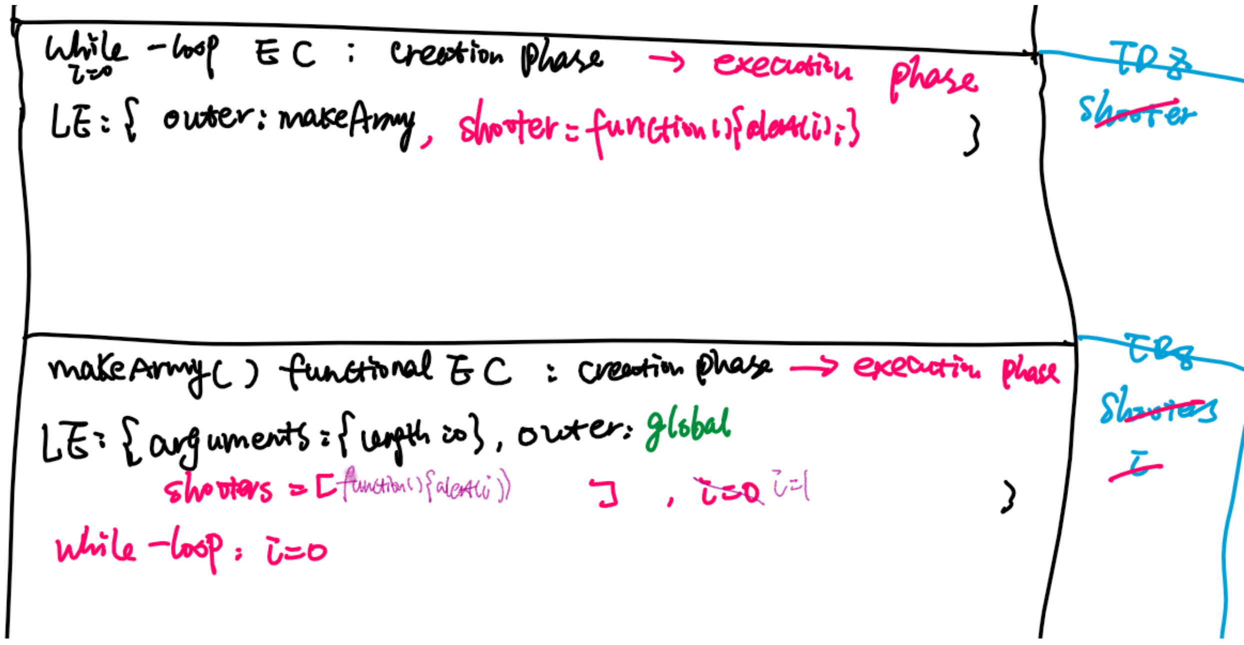


  - executon phase

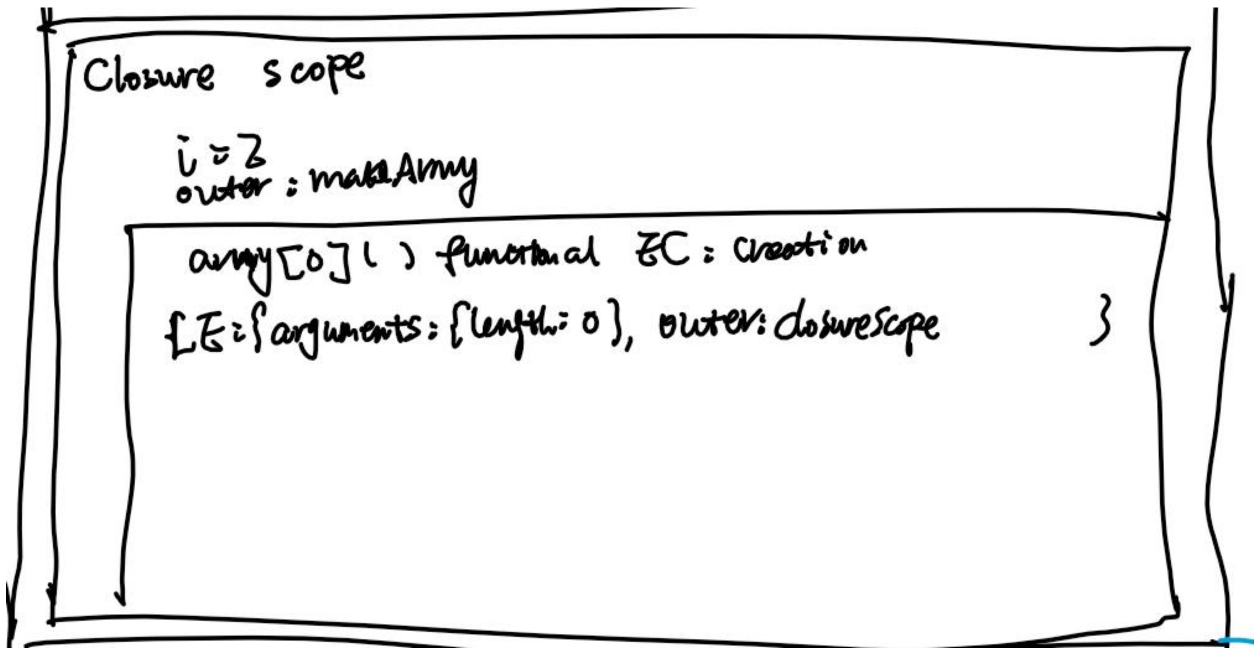
- `shooters.push(shooter);`

- `i++;`

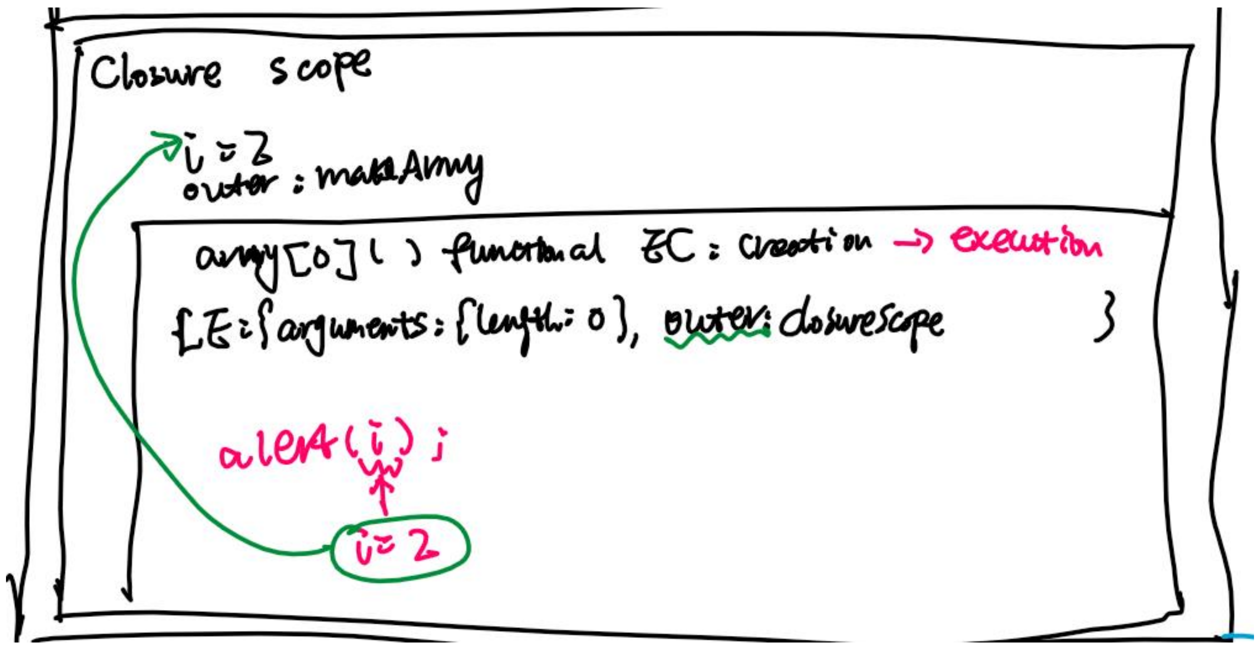
  The two statements above will cause the changes in makeArmy() functional EC


while -loop EC : creation phase → execution phase
i=0
LE: { outer: makeArmy, shooter = function(){alert(i);} }
makeArmy( ) functional EC : creation phase → execution phase
LE: { arguments: {length :0}, outer: global
shooters = [function(){alert(i)}] , ~~i=0~~ i=1 }
while -loop : i=0
~~TDZ~~
~~shooter~~
~~TDZ~~
~~shooters~~
~~i~~

- LE for army[0]()
  - creation phase


Closure scope
i = 3
outer : makeArmy
army[0]( ) functional EC : creation
LE: {arguments: {length: 0}, outer: closureScope }

  - executon phase


Closure scope
i = 2
outer : makeArmy
army[0]( ) functional EC : creation → execution
{LE: {arguments: {length: 0}, outer: closureScope }
alert(i);
i = 2

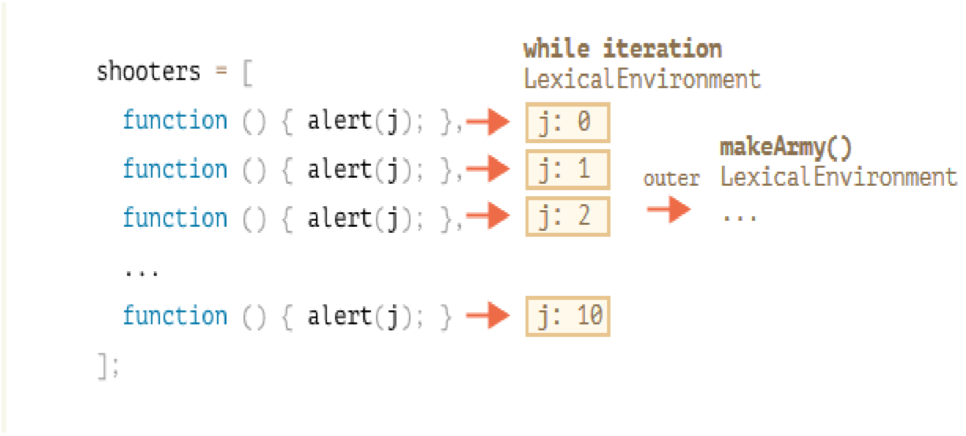
- What will army[0]() alert?

  2

- Can you fix the code?

```
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.forEach(f => f());
```

- How will the diagram change?


shooters = [
function () { alert(j); },
function () { alert(j); },
function () { alert(j); },
...
function () { alert(j); }
];
while iteration
LexicalEnvironment
j: 0
j: 1
j: 2
j: 10
outer
makeArmy()
LexicalEnvironment
...

- Question 2: Write a function printNumbers(from, to) that outputs a number every second, starting from from and ending with to.

Using `setInterval`:

```
function printNumbers(from, to) {
  let current = from;

  let timerId = setInterval(function() {
    alert(current);
    if (current == to) {
      clearInterval(timerId);
    }
    current++;
  }, 1000);
}

// usage:
printNumbers(5, 10);
```

- Question 3:

What will setTimeout show?

importance: 5

In the code below there's a `setTimeout` call scheduled, then a heavy calculation is run, that takes more than 100ms to finish.

When will the scheduled function run?

1. After the loop.
2. Before the loop.
3. In the beginning of the loop.

What is `alert` going to show?

```
let i = 0;

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

// assume that the time to execute this function is >100ms
for(let j = 0; j < 100000000; j++) {
  i++;
}
```

Solution:

Any `setTimeout` will run only after the current code has finished.

The `i` will be the last one: `100000000`.

```
let i = 0;

setTimeout(() => alert(i), 100); // 100000000

// assume that the time to execute this function is >100ms
for(let j = 0; j < 100000000; j++) {
  i++;
}
```