17.1 Closures

Block 1

```
var b = 1;
function someFunction(number) {
function otherFunction(input) {
return b;
}
b= 5;
return otherFunction;
}
var firstResult = someFunction(9);
var result = firstResult(2);
```

This program doesn't output anything because nothing is passed to the console. The program start with declaring a variable `b` and assigning 1 to it. The function `someFunction` changes the value of `b` and returns the function `otherFunction` which returns `b` in the end. So, `firstResult` holds a refernce to `otherFunction` and `result` returns the value of it.

Block 2

```
var a = 1;
function b2() {
a = 10;
return;
function a() { }
}
b2();
console.log(a);
```

The program outputs 1. Since `a` is already declared in the global scope and then locally in the function scope, when it's called outside of the function, the global value of it is returned.

Block 3

```
let i;
for (i = 0; i < 3; i++) {
const log = () => {
console.log(i);
}
setTimeout(log, 100);
}
```

The program returns the `id` of the timer used by `setTimeOut` and outputs the number 3, three times. This is because `setTimeOut` executes the function after the for-loop is finished, and at that point, the value of `i` would be 3. And because of the closure of Javascript, the value of `i` is preserved in the function on the time of execution.