Analiza

Intelligence - Cyber - Data

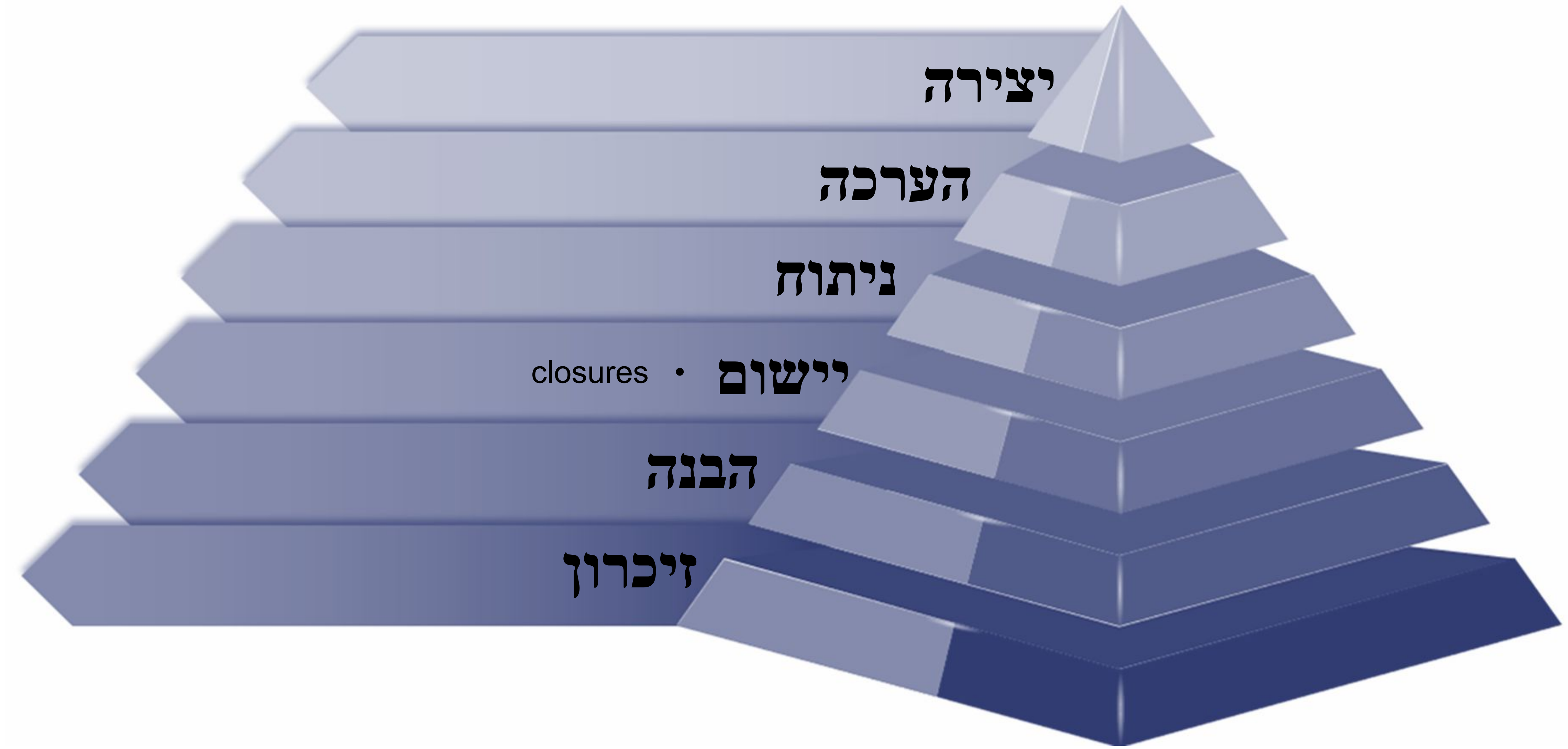
KODCODE

closure

תוכן העניינים

- Goal: understand js array methods
- Topics:
  - Closure
  - factories

מטרות השיעור



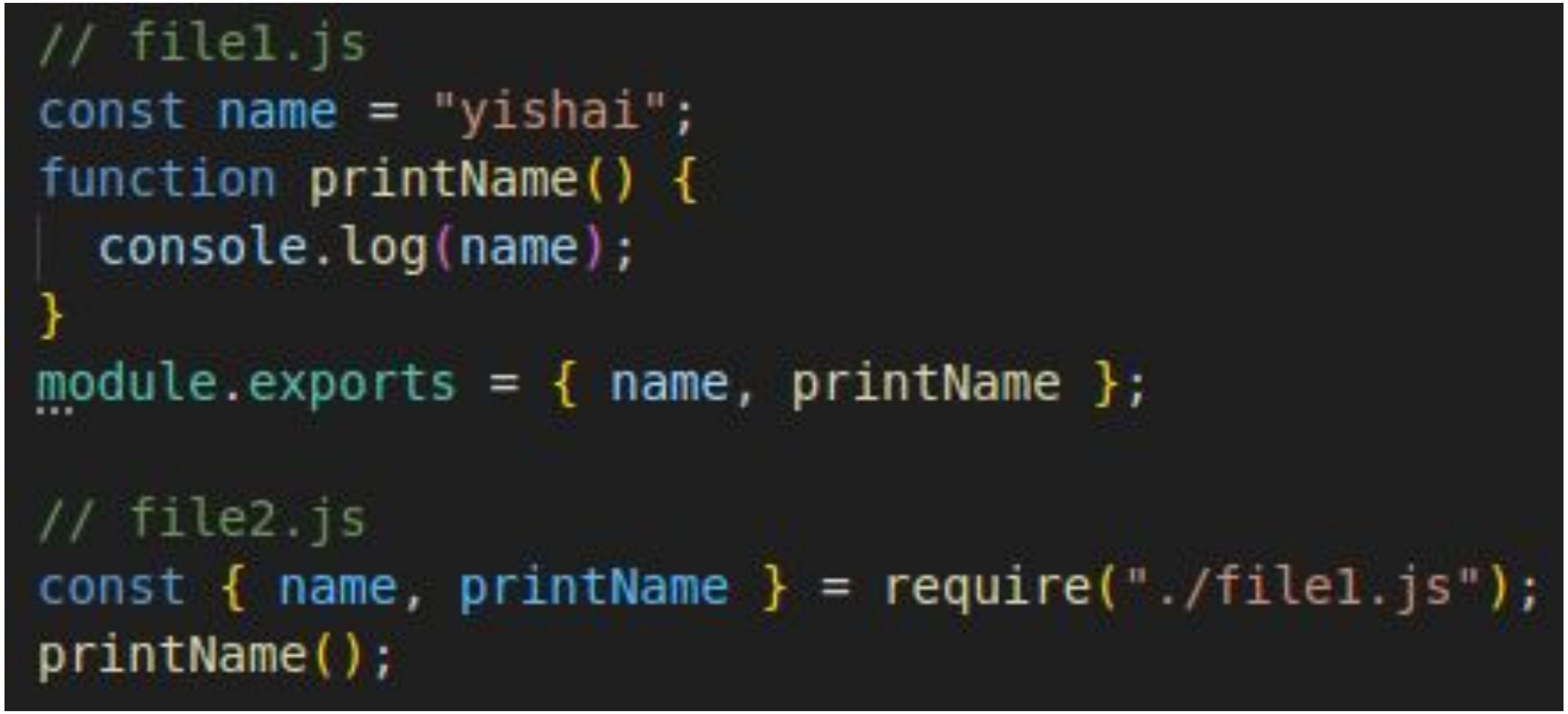
Analiza
Intelligence - Cyber - Data

KODCODE

Closure

Closure

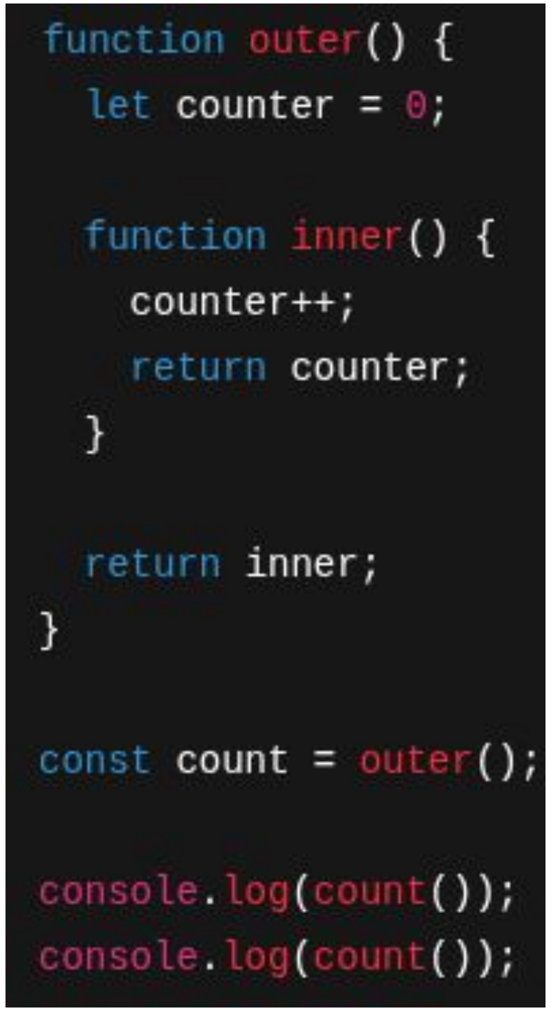
- consider this code: what will happen? why?



```
// file1.js
const name = "yishai";
function printName() {
  console.log(name);
}
module.exports = { name, printName };

// file2.js
const { name, printName } = require("./file1.js");
printName();
```

Closure

- How can a function in another file, can use a variable in another file?
- consider this code:
- what will happen? how?



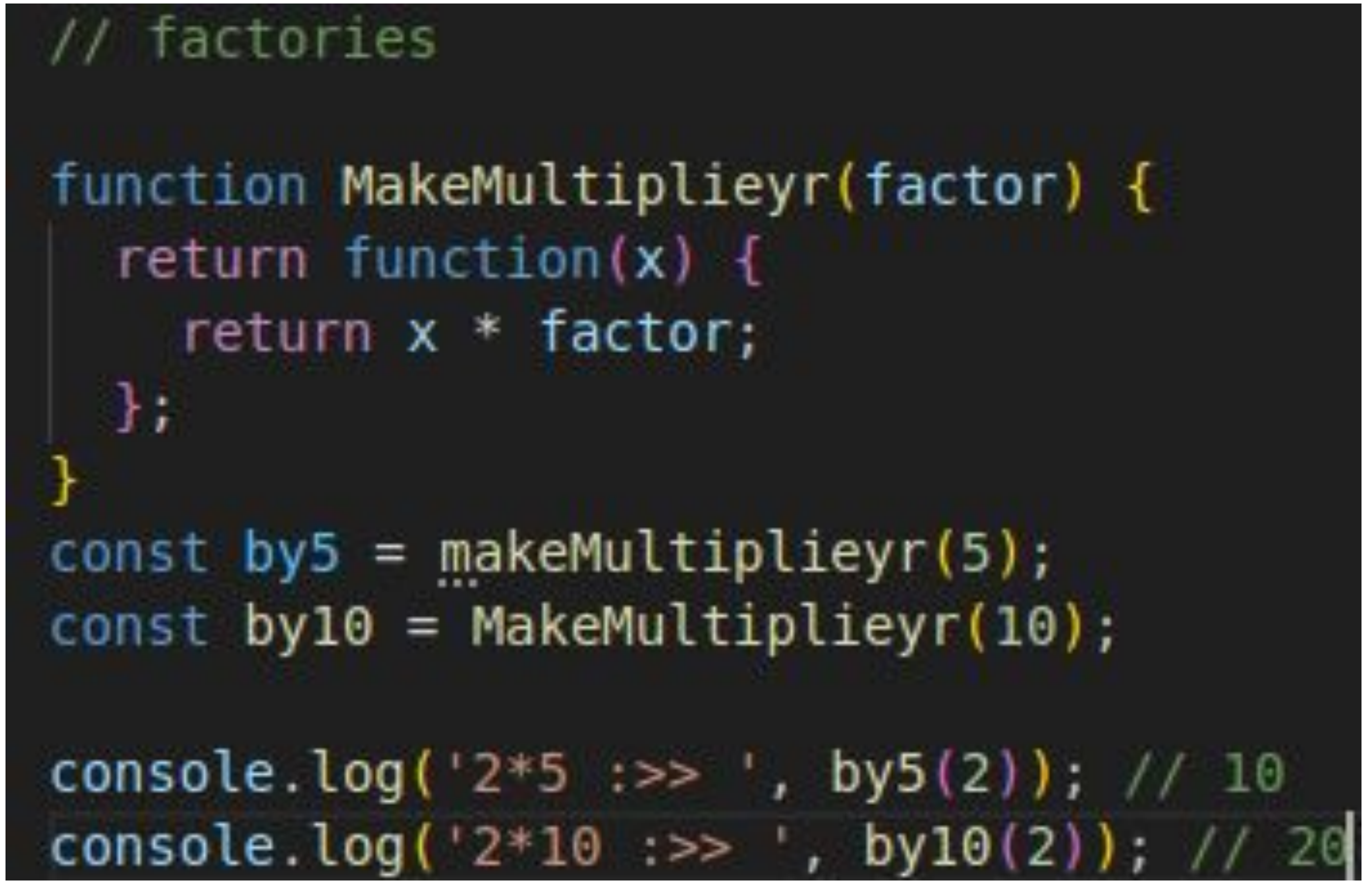
```
function outer() {
  let counter = 0;

  function inner() {
    counter++;
    return counter;
  }

  return inner;
}

const count = outer();

console.log(count());
console.log(count());
```

Closure

- the answer - **closure**.
- Closure is the “memory” of a function.
- If a variable was referenced in the function **declaration**, the function will “remember” it, and can use it later, where ever it will be called.

Closure

- what this can be used for?
- factories!
- consider this:



```
// factories

function MakeMultiplieyr(factor) {
  return function(x) {
    return x * factor;
  };
}
const by5 = makeMultiplieyr(5);
const by10 = MakeMultiplieyr(10);

console.log('2*5 :>> ', by5(2)); // 10
console.log('2*10 :>> ', by10(2)); // 20
```

Closure

- Task:
  - make a spell factory. use closure.
  - each spell you create will have a name.
  - the factory will attach a power level to each spell.
  - each time you create a new spell - you add a bigger and bigger power level. 1,2,3 etc.

Closure

- Task:
  - each spell prints its name and its power level.
  - example:

```
const fireball = spellFactory(“fireball”);
fireball(); // “fireball level 1!”
const advancedFireball = spellFactory(“adFireball”)
advancedFireball() // “fireball level 2!”
```

Closure

- Task 2:
  - Create a factory function createSession(username) that generates session objects.
  - Each session should have a private token string, and support specific operations without exposing the token directly.

Closure

- Task 2:
  - signature: function createSession(username)
  - The returned object should support these methods:
    - getUsername() → returns the username
    - getTokenPreview() → returns the first 4 characters of the token (as a preview)
    - validateToken(t) → returns true only if the given string t matches the internal token
    - regenerateToken() → changes the token to a new random 16-character string

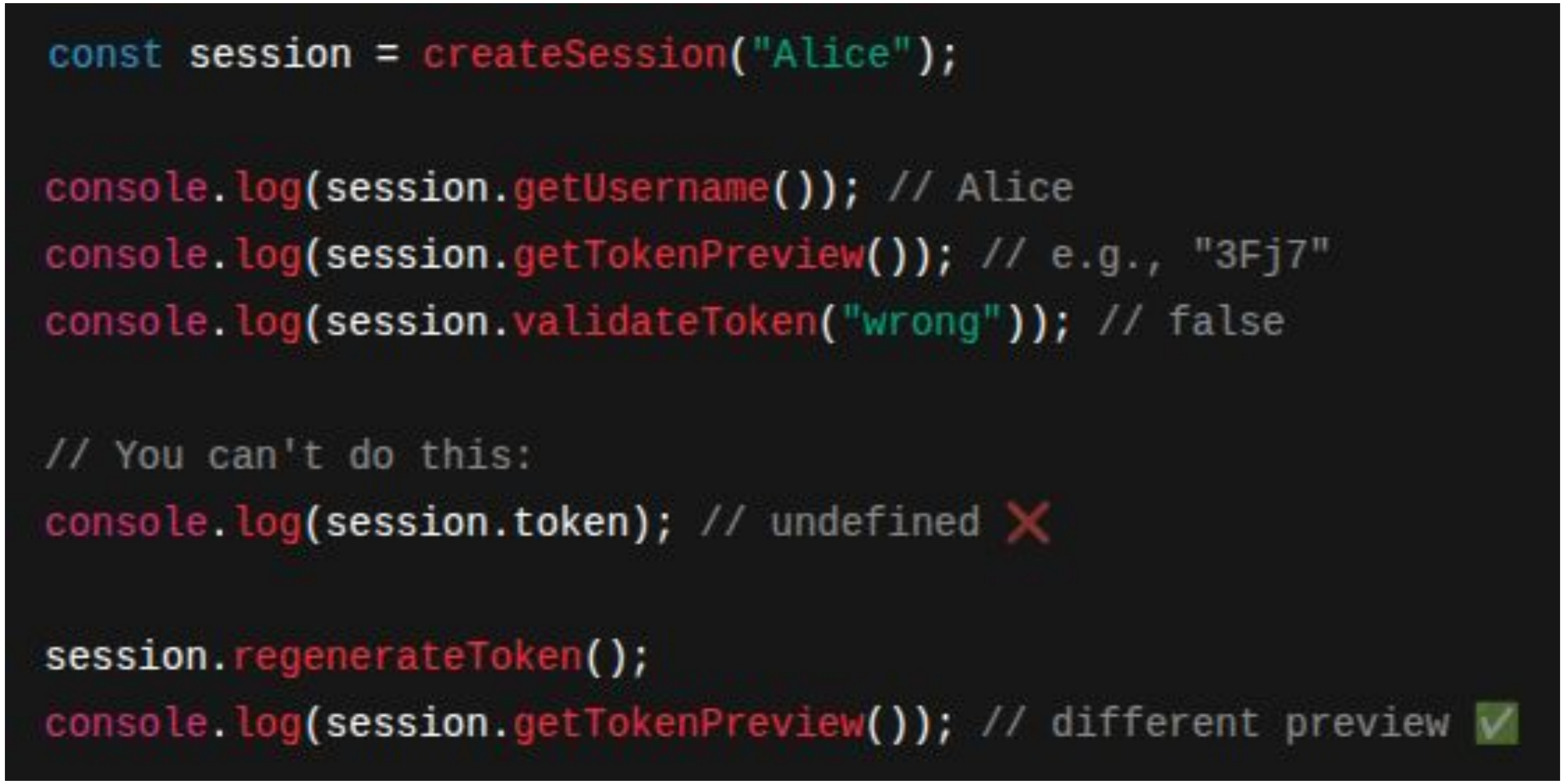
Closure

- Task 2:
  - The token must:
    - be a randomly generated string of 16 alphanumeric characters
    - never be accessible outside the object (only via the above methods)

Closure

- Task 2:
  - Constraints
    - Use closure to store the token
    - Do not use any global variables
    - The token must be regenerated when regenerateToken() is called

Closure

- Task 2:
  - 

```
const session = createSession("Alice");

console.log(session.getUsername()); // Alice
console.log(session.getTokenPreview()); // e.g., "3Fj7"
console.log(session.validateToken("wrong")); // false

// You can't do this:
console.log(session.token); // undefined ❌

session.regenerateToken();
console.log(session.getTokenPreview()); // different preview ✅
```