Scope & Context

DEADLINE: 21.07.2019

FOLDER STRUCTURE

```
FL11_HW10/*                                  *   – required
  └─ homework/*
       └─ index.html*
       └─ .eslintrc.js*
       └─ js*
         └ fighters_game.js*
```

TASK

1) Create a Fighter:

You should create a function or class **'Fighter'** which takes an object with fighter properties and returns interface with fighter methods:

```
        const myFighter = new Fighter({name: 'John', damage: 20, hp: 100, agility: 
25}); // returns an object with methods
```

```
        Note that none of the Fighter's properties may be available directly. The 
only way to get or change each property is to use one of Fighter's methods.
            let name = myFighter.name;
            console.log(name); // undefined
```

Fighter methods:

getName. This function returns fighter's name property.

```
let name = myFighter.getName();
console.log(name); // John
```

getDamage. This function returns fighter's damage property.

```
let damage = myFighter.getDamage();
console.log(damage); // 20
```

getAgility. This function returns fighter's agility property.

```
let agility = myFighter.getAgility();
console.log(agility); // 25
```

getHealth. This function returns fighter's current HP property.

```
let health = myFighter.getHealth();
console.log(health); // 100
```

attack. This function takes argument (instance of 'Fighter'), which will be a defender. Then it randomly calculates if current attack is successful (probability of success is inversely proportional to defender's agility property). For example, if defender's agility is 20, success probability of current attack will be 80%. If defender's agility is 70, success probability of current attack will be 30% etc. If attack is successful, defenders' current HP property is decreased by number of points equal to attacker's damage property and message about successful attack is logged in console. Otherwise, message about missed attack is logged.

```
myFighter.attack(myFighter2);
// John make 20 damage to Jack
myFighter2.attack(myFighter);
// Jack attack missed
```

logCombatHistory. This function logs to console information about fighter's combat history.

```
myFighter.logCombatHistory(); // Name: John, Wins: 0, Losses: 0
```

heal. This function takes amount of health points and add this amount to fighter's current HP (if result is higher than fighter's total HP, than it heals only to total HP).

```
myFighter.heal(50);
```

dealDamage. This function takes amount of health points and reduces these amount from fighter's current HP (if it results to a negative number, current HP should equal 0):

```
myFighter.dealDamage(20);
```

addWin. This function increases fighter's wins property by one.

```
myFighter.addWin();
```

addLoss. This function increases fighter's losses property by one.

```
myFighter.addLoss();
```

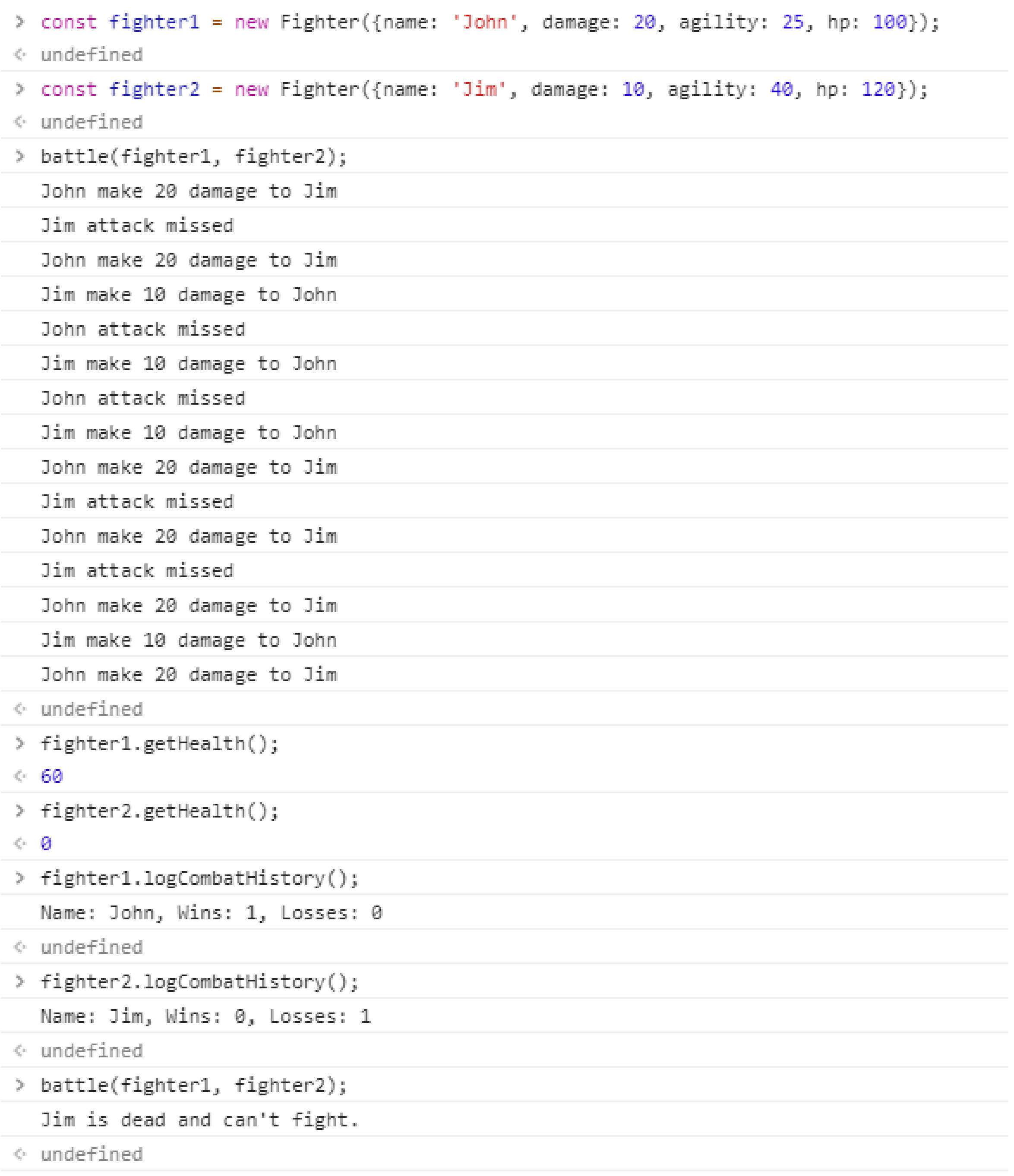
2) Create a 'battle' function:

This function takes 2 arguments (instances of 'Fighter') and simulates battle between them. It performs attacks of each fighter on another until one of them is dead (current HP is 0). After that it increases winner's 'wins' property and loser's 'losses' property by one.

If at the start of battle one of given fighter's is dead (his HP equal to 0), battle shouldn't be simulated and warning message about it should be logged in console.

CODE

Battle example:



```
> const fighter1 = new Fighter({name: 'John', damage: 20, agility: 25, hp: 100});
< undefined
> const fighter2 = new Fighter({name: 'Jim', damage: 10, agility: 40, hp: 120});
< undefined
> battle(fighter1, fighter2);
  John make 20 damage to Jim
  Jim attack missed
  John make 20 damage to Jim
  Jim make 10 damage to John
  John attack missed
  Jim make 10 damage to John
  John attack missed
  Jim make 10 damage to John
  John make 20 damage to Jim
  Jim attack missed
  John make 20 damage to Jim
  Jim attack missed
  John make 20 damage to Jim
  Jim make 10 damage to John
  John make 20 damage to Jim
< undefined
> fighter1.getHealth();
< 60
> fighter2.getHealth();
< 0
> fighter1.logCombatHistory();
  Name: John, Wins: 1, Losses: 0
< undefined
> fighter2.logCombatHistory();
  Name: Jim, Wins: 0, Losses: 1
< undefined
> battle(fighter1, fighter2);
  Jim is dead and can't fight.
< undefined
```

BEFORE SUBMIT

- In order to use npm package manager install nodejs (https://nodejs.org/)
- Install eslint to check your code (npm install -g eslint)
- - open a terminal(or cmd)
- - go to src folder
- - run eslint (i.e. eslint credits_handler.js)
- Code should be without 'errors'
- Code should be clean, readable and well formatted
- Remove all redundant comments
- Test all your functions

SUBMIT

- The folder should be uploaded to github repository ‘**FL11**’ into **master** branch

USEFUL LINKS

- https://css-tricks.com/javascript-scope-closures/
- https://developer.mozilla.org/uk/docs/Web/JavaScript/Reference/Classes