Exercise: This

1. Company

```
class Company {
    // TODO: implement this class...
}
```

Your Task

Write a Company class, which supports the described functionality below.

Functionality

Constructor()

Should have this **1** property:

- **departments** - empty array

addEmployee({username}, {Salary}, {Position}, {Department})

This function should add a new employee to the department with the given name.

- If one of the passed parameters is empty string (""), undefined or null, this function should throw an error with the following message:

"Invalid input!"

- If salary is less than 0, this function should throw an error with the following message:

" Invalid input!"

- If the new employee is hired successfully, you should add him into the departments array and return the following message:

" New employee is hired. Name: {name}. Position: {position}"

bestDepartment()

This **function** should print the department with the highest average salary and its employees sorted by their salary by descending and by name in the following format:

" Best Department is: {best department's name}
Average salary: {best department's average salary}
{employee1} {salary} {position}
{employee2} {salary} {position}
{employee3} {salary} {position}
. . ."

Submission

Submit only your **Company class.**

Examples

This is an example how the code is **intended to be used**:

Sample code usage

`let c = new Company();`
`c.addEmployee("Stanimir", 2000, "engineer", "Construction");`
`c.addEmployee("Pesho", 1500, "electrical engineer", "Construction");`
`c.addEmployee("Slavi", 500, "dyer", "Construction");`
`c.addEmployee("Stan", 2000, "architect", "Construction");`
`c.addEmployee("Stanimir", 1200, "digital marketing manager", "Marketing");`
`c.addEmployee("Pesho", 1000, "graphical designer", "Marketing");`
`c.addEmployee("Gosho", 1350, "HR", "Human resources");`
`console.log(c.bestDepartment());`

Corresponding output
Best Department is: Construction **Average salary: 1500.00** **Stan 2000 architect** **Stanimir 2000 engineer** **Pesho 1500 electrical engineer** **Slavi 500 dyer**

2. Fibonacci

Write a JS function that when called, returns the next Fibonacci number, starting at 0, 1. Use a **closure** to keep the current number.

Input

There will be no input.

Output

The **output** must be a Fibonacci number and must be **returned** from the function.

Examples

Sample execution

`let fib = getFibonator();`
`console.log(fib()); // 1`
`console.log(fib()); // 1`
`console.log(fib()); // 2`
`console.log(fib()); // 3`
`console.log(fib()); // 5`
`console.log(fib()); // 8`
`console.log(fib()); // 13`

3. HEX

```
class Hex {
    // TODO: implement this class...
}
```

Your Task

Write a Hex class, which supports the described functionality below.

Functionality

Constructor({value})

Should have this **1** property:

- **`value`** - number

valueOf()

This Function Should Return the Value Property of the Hex Class.

toString()

This **function** will show its hexadecimal value starting with "0x"

plus({number})

This function should add a number or Hex object and return a new Hex object.

minus({number})

This function should subtract a number or Hex object and return a new Hex object.

parse({string})

Create a `parse` class method that can **parse** Hexadecimal numbers and convert them to standard decimal numbers.

Submission

Submit only your **Hex class.**

Examples

This is an example how the code is **intended to be used**:

Sample execution

`let FF = new Hex(255);`
`console.log(FF.toString());`
`FF.valueOf() + 1 == 256;`
`let a = new Hex(10);`
`let b = new Hex(5);`
`console.log(a.plus(b).toString());`
`console.log(a.plus(b).toString()==='0xF');`

0xFF **0xF** **True**