Classes - Exercises

1. Cats 1
2. Person 2
3. Circle 2
4. Point Distance 3
5. Class Laptop 3
6. School Book 4
7. Rectangle 6
8. Data Class 7
9. Tickets 7
10. Sorted List 9
11. String Container 9
12. Company 10
13. Car Company 11

1. Cats

Write a function that receives **array** of strings in the following format **'{cat name} {age}'**.

Create a **Cat class** that receives in the **constructor** the **name** and the **age** parsed from the input.

It should also have a method named **"meow"** that will print **"{cat name}, age {age} says meow"** on the console.

For each of the strings provided, you must **create a cat object** and invoke the **.meow()** method**.**

Examples

Input	**Output**
['Mellon 2', 'Tom 3**']**	**Mellon, age 2 says meow** **Tom, age 3 says meow**
['Branch **1', 'Poppy 3', '**Goldy **2']**	**Branch, age 1 says meow** **Poppy, age 3 says meow** **Goldy, age 2 says meow**

- Create a **Cat class** with properties and methods described above

- Parse the input data
- Create all objects using the class constructor and the parsed input data, store them in an array
- Loop through the array using **for…of** a cycle and **invoke .meow()** method

2. Person

Write a **class** that represents a personal record. It has the following properties, all set from the constructor:

- **firstName**
- **lastName**
- **age**
- **email**

And a method **toString()**, which prints a summary of the information. See the example for formatting details.

The **toString()**method should **return** a string in the following format:

`{firstName} {lastName} (age: {age}, email: {email})`

Sample Input

let person = new Person('Homer', 'Simpson', 42, 'homer@yahoo.com');
console.log(person.toString());

Output
Homer Simpson (age: 42, email: homer@yahoo.com)

3. Circle

Write a **class** that represents a **Circle**. It has only one data property - its **radius**, and it is set through the **constructor**. The class needs to have **getter** and **setter** methods for its **diameter** - the setter needs to calculate the radius and change it and the getter needs to use the radius to calculate the diameter and return it.

The circle also has a getter **area()**, which calculates and **returns** its area.

The **diameter()** and **area()** getters should **return** numbers.

Sample Input	**Output**

let c = new Circle(2);
console.log(`Radius: ${c.radius}`);
	
	Radius: 2

console.log(`Diameter: ${c.diameter}`);
console.log(`Area: ${c.area}`);
c.diameter = 1.6;
console.log(`Radius: ${c.radius}`);
console.log(`Diameter: ${c.diameter}`);
console.log(`Area: ${c.area}`);
	Diameter: 4
	Area: 12.566370614359172
	Radius: 0.8
	Diameter: 1.6
	Area: 2.0106192982974678

4. Point Distance

Write a **class** that represents a **Point**. It has **x** and **y** coordinates as properties, that are set through the constructor, and a **static method** for finding the distance between two points, called **distance()**.

The **distance()** method should receive two **Point** objects as parameters.

The **distance()** method should **return** a number, the distance between the two-point parameters.

Sample Input	**Output**

let p1 = new Point(5, 5);
let p2 = new Point(9, 8);
console.log(Point.distance(p1, p2));
	5

5. Class Laptop

Create a **class Laptop** that has the following properties:

- **info** – object that contains:
    - **producer** – string
    - **age** – number
    - **brand** – string
- **isOn** – boolean (false by default)
- **turnOn** – a function that **sets the isOn** variable to **true**
- **turnOff** – a function that **sets the isOn** variable to **false**
- **showInfo** – a function that returns the **producer, age, and brand as JSON**
- **quality** – number (every time the laptop **is turned on/off the quality decreases by 1**)
- **getter price** – number (**800 – {age * 2} + (quality * 0.5)**)

The **constructor** should receive the **info as an object and the quality.**

Input	Output

let info = {producer: "Asus", age: 2, brand: "Zenbook"}
let laptop = new Laptop(info, 10)
laptop.turnOn()
console.log(laptop.showInfo())
laptop.turnOff()
console.log(laptop.quality)
laptop.turnOn()
console.log(laptop.isOn)
console.log(laptop.price)
	{"producer":"Asus","age":2,"brand":"Zenbook"}
	8
	true
	799.5

let info = {producer: "Lenovo", age: 1, brand: "Legion"}
let laptop = new Laptop(info, 10)
laptop.turnOn()
console.log(laptop.showInfo())
laptop.turnOff()
laptop.turnOn()
laptop.turnOff()
console.log(laptop.isOn)
	{"producer":"Lenovo","age":1,"brand":"Legion"}
	false

6. School Book

Arrange all students by **grade**. Process students and store them into a school register before the new school year hits. As a draft, you have a list of all the students from **last year** but mixed. Keep in mind that if a student has a lower score than 3, he does not go into the next class. As a result of your work, you have to print the entire school register **sorted** in **ascending order by grade** already filled with all the students from last year in the format:

`{nextGrade} Grade

List of students: {All students in that grade}

Average annual score from last year: {average annual score on the entire class from last year}`

Delimiter row **{===}**

The input will be an **array** with strings, each containing a student's name, last year's grade, and an annual score. The **average annual score from last year** should be **formatted to the second decimal point**.

Input	Output
[**"Student name: Mark, Grade: 8, Graduated with an average score: 4.75",** **"Student name: Ethan, Grade: 9, Graduated with an average score: 5.66",** **"Student name: George, Grade: 8, Graduated with an average score: 2.83",** **"Student name: Steven, Grade: 10, Graduated with an average score: 4.20",** **"Student name: Joey, Grade: 9, Graduated with an average score: 4.90",** **"Student name: Angus, Grade: 11, Graduated with an average score: 2.90",** **"Student name: Bob, Grade: 11, Graduated with an average score: 5.15",** **"Student name: Daryl, Grade: 8, Graduated with an average score: 5.95",** **"Student name: Bill, Grade: 9, Graduated with an average score: 6.00",** **"Student name: Philip, Grade: 10, Graduated with an average score: 5.05",** **"Student name: Peter, Grade: 11, Graduated with an average score: 4.88",**	9 Grade List of students: Mark, Daryl Average annual score from last year: 5.35 === 10 Grade List of students: Ethan, Joey, Bill Average annual score from last year: 5.52 === 11 Grade List of students: Steven, Philip, Gavin Average annual score from last year: 4.42 === 12 Grade List of students: Bob, Peter Average annual score from last year: 5.02 ===

"Student name: Gavin, Grade: 10, Graduated with an average score: 4.00" **]**	
[**'Student name: George, Grade: 5, Graduated with an average score: 2.75',** **'Student name: Alex, Grade: 9, Graduated with an average score: 3.66',** **'Student name: Peter, Grade: 8, Graduated with an average score: 2.83',** **'Student name: Boby, Grade: 5, Graduated with an average score: 4.20',** **'Student name: John, Grade: 9, Graduated with an average score: 2.90',** **'Student name: Steven, Grade: 2, Graduated with an average score: 4.90',** **'Student name: Darsy, Grade: 1, Graduated with an average score: 5.15'** **]**	2 Grade List of students: Darsy Average annual score from last year: 5.15 === 3 Grade List of students: Steven Average annual score from last year: 4.90 === 6 Grade List of students: Boby Average annual score from last year: 4.20 === 10 Grade List of students: Alex Average annual score from last year: 3.66 ===

7. Rectangle

Write a **class Rectangle** for a rectangle object. It needs to have a **width** (Number), **height** (Number), and **color** (String) properties, which are set from the constructor, and a **calcArea()** method, that calculates and **returns** the rectangle's area.

The **calcArea()** method should **return** a number.

Sample Input	Output
let rect = new Rectangle(4, 5, 'Red');	**4**
console.log(rect.width);	**5**
console.log(rect.height);	**Red**

console.log(rect.color);
console.log(rect.calcArea());
	20

8. Data Class

Write a **class Request** that holds data about an HTTP request. It has the following properties:

- **method** (String)
- **uri** (String)
- **version** (String)
- **message** (String)
- **response** (String)
- **fulfilled** (Boolean)

The first four properties (**method**, **uri**, **version**, **message**) are set through the **constructor**, in the listed order. The **response** property is initialized to **undefined** and the **fulfilled** property is initially set to **false**.

Sample Input	Resulting object
let myData = new Request('GET', 'http://google.com', 'HTTP/1.1', '') **console.log(myData);**	**Request {** **method: 'GET',** **uri: 'http://google.com',** **version: 'HTTP/1.1',** **message: '',** **response: undefined,** **fulfilled: false** **}**

9. Tickets

Write a program that manages a database of tickets. A ticket has a **destination,** a **price,** and a **status**. Your program will receive **two arguments** - the first is an **array of strings** for ticket descriptions and the second is a **string**, representing a **sorting criterion**. The ticket descriptions have the following format:

<destinationName>|<price>|<status>

Store each ticket and at the end of execution **return** a sorted summary of all tickets, sorted by either **destination**, **price,** or **status**, depending on the **second parameter** that your program received. Always sort in ascending order (the default behavior for **alphabetical** sort). If two tickets compare the same, use order of insertion.

Return a **sorted array** of all the tickets that were registered.

Sample Input	Output Array
['Philadelphia|94.20|available', 'New York City|95.99|available', 'New York City|95.99|sold', 'Boston|126.20|departed'], 'destination'	**[Ticket { destination: 'Boston', price: 126.20, status: 'departed' }, Ticket { destination: 'New York City', price: 95.99, status: 'available' }, Ticket { destination: 'New York City', price: 95.99, status: 'sold' }, Ticket { destination: 'Philadelphia', price: 94.20, status: 'available' }]**
['Philadelphia|94.20|available', 'New York City|95.99|available', 'New York City|95.99|sold', 'Boston|126.20|departed'], 'status'	**[Ticket { destination: 'Philadelphia', price: 94.20, status: 'available' }, Ticket { destination: 'New York City', price: 95.99, status: 'available' }, Ticket { destination: 'Boston', price: 126.20, status: 'departed' }, Ticket { destination: 'New York City', price: 95.99, status: 'sold' }]**

10. Sorted List

Implement a **class List**, which **keeps** a list of numbers, sorted in **ascending order**. It must support the following functionality:

- **add(element)** - adds a new element to the collection
- **remove(index)** - removes the element at position **index**

- **get(index)** - returns the value of the element at position **index**
- **size** - number of elements stored in the collection

The **correct order** of the elements must be kept **at all times**, regardless of which operation is called. **Removing** and **retrieving** elements **shouldn't work** if the provided index points **outside the length** of the collection (either throw an error or do nothing). Note the **size** of the collection is **not** a function.

Sample Input	Output
let list = new List();	**6**
list.add(5);	**7**
list.add(6);	
list.add(7);	
console.log(list.get(1));	
list.remove(1);	
console.log(list.get(1));	

11. String Container

Create a class **StringContainer**, which holds the **single string** and a **length** property. The class should be initialized with a **string** and an **initial length.** The class should always keep the **initial state** of its **given string**.

Name the two properties **innerString** and **innerLength**.

There should also be functional for increasing and decreasing the initial **length** property. Implement function **increase(length)** and **decrease(length)**, which manipulate the length property with the **given value**.

The length property is **a numeric value** and should not fall below **0**. It should not throw any errors, but if an attempt to decrease it below 0 is done, it should be automatically set to **0**.

You should also implement functionality for **toString()** function, which returns the string, the object was initialized with. If the length of the string is greater than the **length property**, the string should be cut from right to left, so that it has the **same length** as the **length property**, and you should add **3 dots** after it if such **truncation** was **done**.

If the length property is **0**, just return **3 dots.**

stringContainer.js

let *test* = new StringContainer("Test", 5);
console.log(*test*.toString()); *// Test*
 	
test.decrease(3);
console.log(*test*.toString()); *// Te...*

```
test.decrease(5);
console.log(test.toString()); // ...

test.increase(4);
console.log(test.toString()); // Test
```

Store the initial string in a property, and do not change it. Upon calling the **toString()** function, truncate it to the **desired value** and return it.

12. Company

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

Write a class **Company**, which following these requirements:

The **constructor** takes no parameters:

You could initialize an object:

- **departments** - empty object

addEmployee({name}, {salary}, {position}, {department})

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

- If one of the passed parameters is an 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** object with the current name of the department and return the following message: **`New employee is hired. Name: {name}. Position: {position}`**

bestDepartment()

This function should return the **department** with the **highest average salary rounded** to the second digit after the decimal point and its **employees sorted** by their **salary** by **descending** order and by their **name** in **ascending** order as a second criterion:

`Best Department is: {best department's name}

Average salary: {best department's average salary}

{employee1} {salary} {position}

{employee2} {salary} {position}

{employee3} {salary} {position}

…`

Sample code usage

```
let c = new Company();
c.addEmployee("Stamat", 2000, "engineer", "Construction");
c.addEmployee("Peter", 1500, "electrical engineer", "Construction");
c.addEmployee("Martin", 500, "cleaner", "Construction");
c.addEmployee("Stanley", 2000, "architect", "Construction");
c.addEmployee("Stamat", 1200, "digital marketing manager", "Marketing");
c.addEmployee("Peter", 1000, "graphical designer", "Marketing");
c.addEmployee("George", 1350, "HR", "Human resources");
console.log(c.bestDepartment());
```

Corresponding output

```
Best Department is: Construction
Average salary: 1500.00
Stanley 2000 architect
Stamat 2000 engineer
Peter 1500 electrical engineer
Martin 500 cleaner
```

13. Car Company

You are tasked to create a register for a company that produces cars. You need to store **how many cars** have been produced from a **specific model** of a **specific brand**.

The **input** comes as array of strings. Each element holds information in the following format:

"**{carBrand} | {carModel} | {producedCars}**"

The **carBrand** and **carModel** are **strings**, the **producedCars** are **numbers**. If the **carBrand** you've received **already exists**, just add the **new carModel** to it with the **produced cars as its value**. If even the **carModel** exists, just **add** the **given value** to the **current one**.

As **output**, you need to print - **for every car brand**, the **car models**, and a **number of cars produced** from that model. The output format is:

```
`{carBrand}
 ###{carModel} -> {producedCars}
 ###{carModel2} -> {producedCars}
 ...`
```

The order of printing is the **order in which the brands and models first appear in the input**. The first brand in the input should be the first printed and so on. For each brand, the first model received from that brand, should be the first printed and so on.

Input	Output
['Audi | Q7 | 1000',	**Audi**
'Audi | Q6 | 100',	**###Q7 -> 1000**
'BMW | X5 | 1000',	**###Q6 -> 100**
'BMW | X6 | 100',	**BMW**
'Citroen | C4 | 123',	**###X5 -> 1000**
'Volga | GAZ-24 | 1000000',	**###X6 -> 100**
'Lada | Niva | 1000000',	**Citroen**
'Lada | Jigula | 1000000',	**###C4 -> 145**
'Citroen | C4 | 22',	**###C5 -> 10**
'Citroen | C5 | 10']	**Volga**
	###GAZ-24 -> 1000000
	Lada
	###Niva -> 1000000
	###Jigula -> 1000000