Lab: Classes

Classes

1. Rectangle

Write a **class** 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.

Input

The constructor function will receive valid parameters.

Output

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

Sample Input	Output

`let rect = new Rectangle(4, 5, 'red');`
`console.log(rect.width);`
`console.log(rect.height);`
`console.log(rect.color);`
`console.log(rect.calcArea());`
	`4`
	`5`
	`Red`
	`20`

What to submit?

You are only required to submit the **Rectangle class**. No need to include the codes from the example above.

Class Signature: class **Rectangle**

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.

Input

The constructor function will receive valid parameters.

Output

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

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

Sample Input
`let person = new Person('Anna', 'Simpson', 22, 'anna@yahoo.com');` `console.log(person.toString());`
Output
`Anna Simpson (age: 22, email: anna@yahoo.com)`

What to submit?

You are only required to submit the **Person class**. No need to include the codes from the example above.

Class Signature: class **Person**

3. Get Persons

Write a function that returns an array of **`Person`** objects. Use the class from the previous task, create the following instances, and return them in an array:

First Name	Last Name	Age	Email
Anna	Simpson	22	anna@yahoo.com
Kingsland University			
Stephan	Johnson	25	
Gabriel	Peterson	24	g.p@gmail.com

For any empty cells, do not supply a parameter (call the constructor with less parameters).

Input / Output

There will be **no input**, the data is static and matches the table above. As **output**, **return an array** with **`Person`** **instances**.

What to submit?

Create a **function main** that returns an array of **Person**.

```
class Person {
  // methods
}

function main() {
  // code
}
```

4. Circle

Write a **class** that represents a **`Circle`**. It has only one data property - it's **`radius`**, and it is set trough 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.

Input

The constructor function and diameter setter will receive valid parameters.

Output

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

Sample Input	Output
`let c = new Circle(2);`	
`console.log(`Radius: ${c.radius}`);`	`2`
`console.log(`Diameter: ${c.diameter}`);`	`4`
`console.log(`Area: ${c.area}`);`	`12.566370614359172`
`c.diameter = 1.6;`	
`console.log(`Radius: ${c.radius}`);`	`0.8`
`console.log(`Diameter: ${c.diameter}`);`	`1.6`
`console.log(`Area: ${c.area}`);`	`2.0106192982974678`

What to submit?

You are only required to submit the **Circle class**. No need to include the codes from the example above.

Class Signature: class **Circle**

5. Point Distance

Write a JS **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()`**.

Input

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

Output

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

Example

Sample Input	Output

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

What to submit?

You are only required to submit the **Point class**. No need to include the codes from the example above.

Class Signature: class **Point**

6. Cards

You need to write an **IIFE** that results in an object containing two properties **`Card`** which is a class and **`Suits`** which is an object that will hold the possible suits for the cards.

The **`Suits`** object should have exactly these 4 properties:

- **SPADES**: ♠
- **HEARTS**: ♥
- **DIAMONDS**: ♦
- **CLUBS**: ♣

Where the key is **SPADES**, **HEARTS** e.t.c. and the value is the actual symbol ♠, ♥ and so on.

The **`Card`** class should allow for creating cards, each card has 2 properties **face** and **suit**. The **valid** faces are the following **`["2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A"]`** any other are considered invalid.

The **`Card`** class should have **`setters`** and **`getters`** for the **face** and **suit** properties, when creating a card or setting a property validations should be performed, if an invalid face or a suit not in the **`Suits`** object is passed an **`Error`** should be **thrown**.

Code Template

You are required to write and submit an **IIFE** which results in an object containing the above-mentioned **`Card`** and **`Suits`** as properties. Here is an example template you can use:

cards.js

```
(function(){

    // TODO:

    return {
        Suits:Suits,
        Card:Card
    }
}())
```

Screenshot

An example usage should look like this:

```
let result = (function(){...}());
let Card = result.Card;
let Suits = result.Suits;

let card = new Card("Q",Suits.CLUBS);
card.face = "A";
card.suit = Suits.DIAMONDS;
let card2 = new Card("1",Suits.DIAMONDS);   //Should throw Error
```

What to submit?

Create a **function main** that returns the **IIFE**.

```
function main() {
  return (function(){
    // code
  }())
}
```