

Lab: Prototypes and Inheritance

Problems for exercises and homework for the ["JavaScript Applications" course @ SoftUni](https://judge.softuni.org/Contests/2770/Prototypes-and-Inheritance-Lab). Submit your solutions in the SoftUni Judge system at <https://judge.softuni.org/Contests/2770/Prototypes-and-Inheritance-Lab>

1. Person

Write a JS program which takes **first** & **last** names as **parameters** and returns an object with **firstName**, **lastName** and **fullName** (`"{firstName} {lastName}"`) properties which should be all **accessible**, we discovered that "accessible" also means "mutable". This means that:

- If **firstName** or **lastName** have changed, then **fullName** should also be changed.
- If **fullName** is changed, then **firstName** and **lastName** should also be changed.
- If **fullName** is **invalid**, you should not change the other properties.
- A **valid full name** is in the format: `"{firstName} {lastName}"`.

Examples

Sample Input

```
let person = new Person("Peter", "Ivanov");
console.log(person.fullName); //Peter Ivanov
person.firstName = "George";
console.log(person.fullName); //George Ivanov
person.lastName = "Peterson";
console.log(person.fullName); //George Peterson
person.fullName = "Nikola Tesla";
console.log(person.firstName); //Nikola
console.log(person.lastName); //Tesla
```

```
let person = new Person("Albert", "Simpson");
console.log(person.fullName); //Albert Simpson
person.firstName = "Simon";
console.log(person.fullName); //Simon Simpson
person.fullName = "Peter";
console.log(person.firstName); // Simon
console.log(person.lastName); // Simpson
```

2. Person and Teacher

Write a class **Person** and a class **Teacher** which extends **Person**.

- The **Person** class should have a **name** and an **email**
- The **Teacher** class should have a **name**, an **email**, and a **subject**

Input \ Output

There will be **NO** input. Your function should return an object containing the classes **Person** and **Teacher**.

Hints:

template.js

```
function personAndTeacher() {
  // TODO:

  return {
    Person,
    Teacher
  }
}
```

3. Inheriting and Replacing ToString

Extend the **Person** and **Teacher** from the previous task and add a class **Student** inheriting from **Person** with additional property **course**. Add **toString()** functions to all classes, the formats should be as follows:

- **Person** - returns "Person (name: {name}, email: {email})"
- **Student** - returns "Student (name: {name}, email: {email}, course: {course})"
- **Teacher** - returns "Teacher (name: {name}, email: {email}, subject: {subject})"

Try to reuse code by using the **toString()** function of the base class.

Input / Output

There will be **NO** input. Your function should return an object containing the classes **Person**, **Teacher** and **Student**.

Hints:

template.js

```
function toStringExtension() {
  // TODO:

  return {
    Person,
    Teacher,
    Student
  }
}
```

4. Extend Prototype

Write a function that receives a **class** and attaches to it a property **species** with the value "**Human**" and a function **toSpeciesString()**. When called, the function returns a string with the format:

"I am a <species>. <toString()>"

The function **toString()** is called from the current instance (call using **this**).

Input / Output

Your function will receive a **class** whose prototype it should extend. There is **NO** output, your function should only attach the properties to the given class' prototype.

template.js

```
function extendPrototype(classToExtend) {  
    // TODO:  
}
```

5. Class Hierarchy

Write a function that returns 3 classes - **Figure**, **Circle** and **Rectangle**.

Figure:

- Should have property units ("m", "cm", "mm") with default value "cm"
- Should have a **getter area**
- Has method **changeUnits** that sets different units for that figure
- **Has method toString**, which returns: ``Figures units: {units}``

Circle:

- Extends **Figure**
- Has a property **radius**
- Overrides **area** getter to return the area of the Circle ($\text{PI} * r * r$)
- **toString()** - should return a string representation of the figure in the format:
``Figures units: {type} Area: {area} - radius: {radius}``

Rectangle:

- Extends **Figure**
- Has properties **width**, **height** and **units** (extended from the class Figure)
- Overrides **area** getter to return the area of the **Rectangle** ($\text{width} * \text{height}$)
- **toString()** - should return a string representation of the figure in the format:
``Figures units: {type} Area: {area} - width: {width}, height: {height}``

Note: All Parameters Passed in the Constructors Are in Centimeters ("cm")

Input / Output

There will be **no** input. Your function should return an object containing the **Figure**, **Circle** and **Rectangle** classes.

Examples

This code demonstrates how your classes should behave:

Sample Code

```
let c = new Circle(5);  
console.log(c.area); // 78.53981633974483  
console.log(c.toString()); // Figures units: cm Area: 78.53981633974483 - radius: 5  
  
let r = new Rectangle(3, 4, 'mm');  
console.log(r.area); // 1200  
console.log(r.toString()); //Figures units: mm Area: 1200 - width: 30, height: 40  
  
r.changeUnits('cm');  
console.log(r.area); // 12
```

```
console.log(r.toString()); // Figures units: cm Area: 12 - width: 3, height: 4  
c.changeUnits('mm');  
console.log(c.area); // 7853.981633974483  
console.log(c.toString()) // Figures units: mm Area: 7853.981633974483 - radius: 50
```