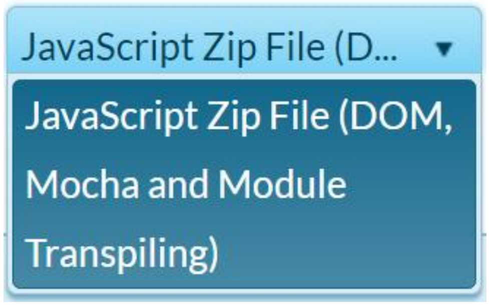
# Lab: Modules, Babel and Transpiling

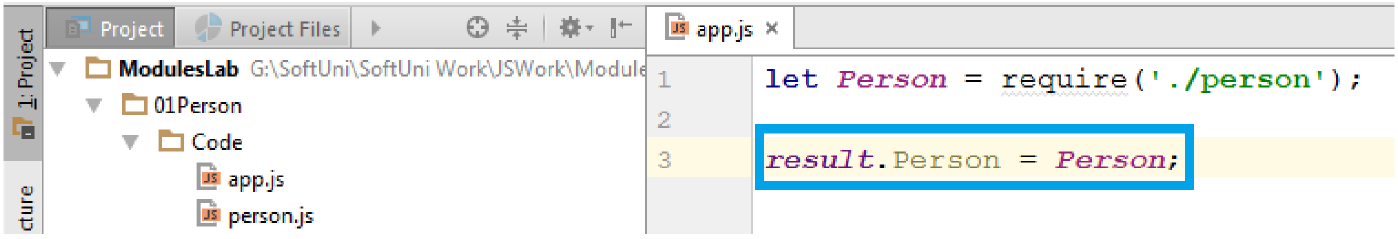
Problems for exercises and homework for the "JavaScript Advanced" course @ SoftUni. Submit your solutions in the SoftUni judge system at https://judge.softuni.bg/Contests/342/.

## Submitting to Judge

For the following tasks, you will be submitting using the **JavaScript Zip File** strategy.


JavaScript Zip File (D...
JavaScript Zip File (DOM, Mocha and Module Transpiling)

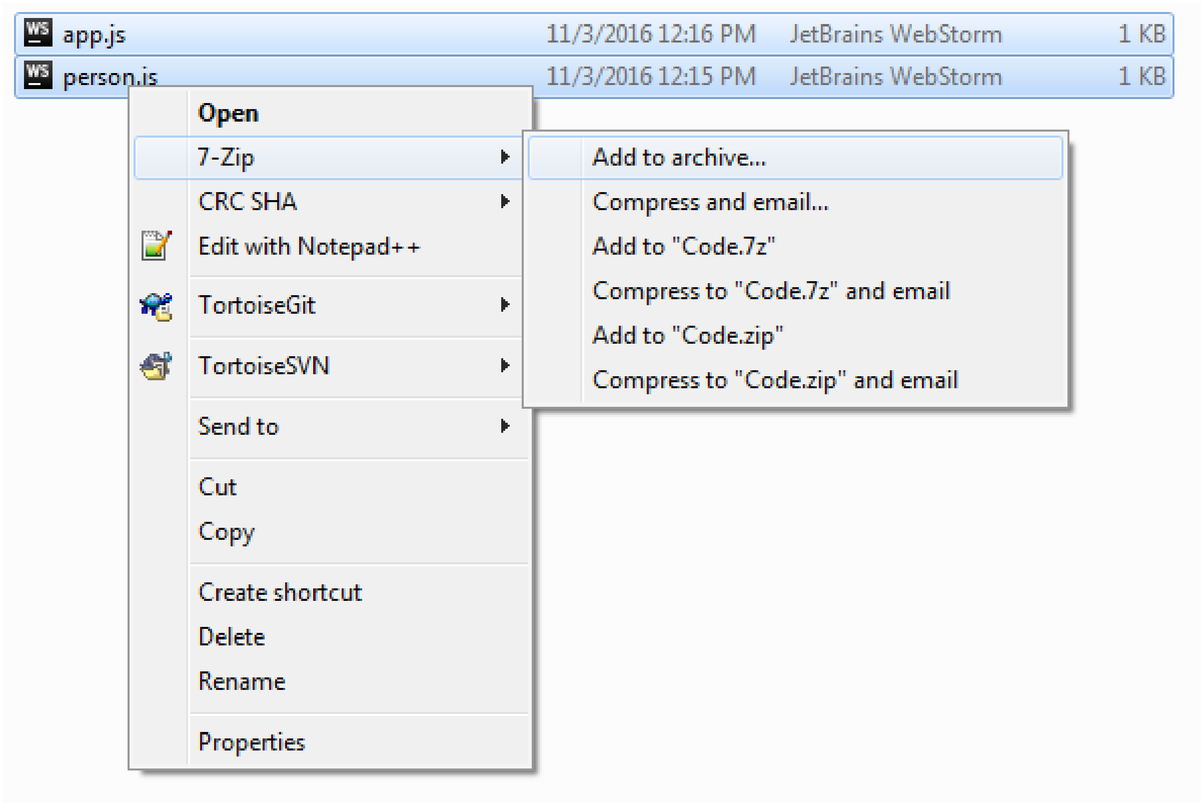
The strategy requires that you submit a **zip** file containing all your **JavaScript** (**.js**) files. Your main file should always be named **`app.js`** - it is the beginning of your program and should **`import`**/**`require`** all other files needed.



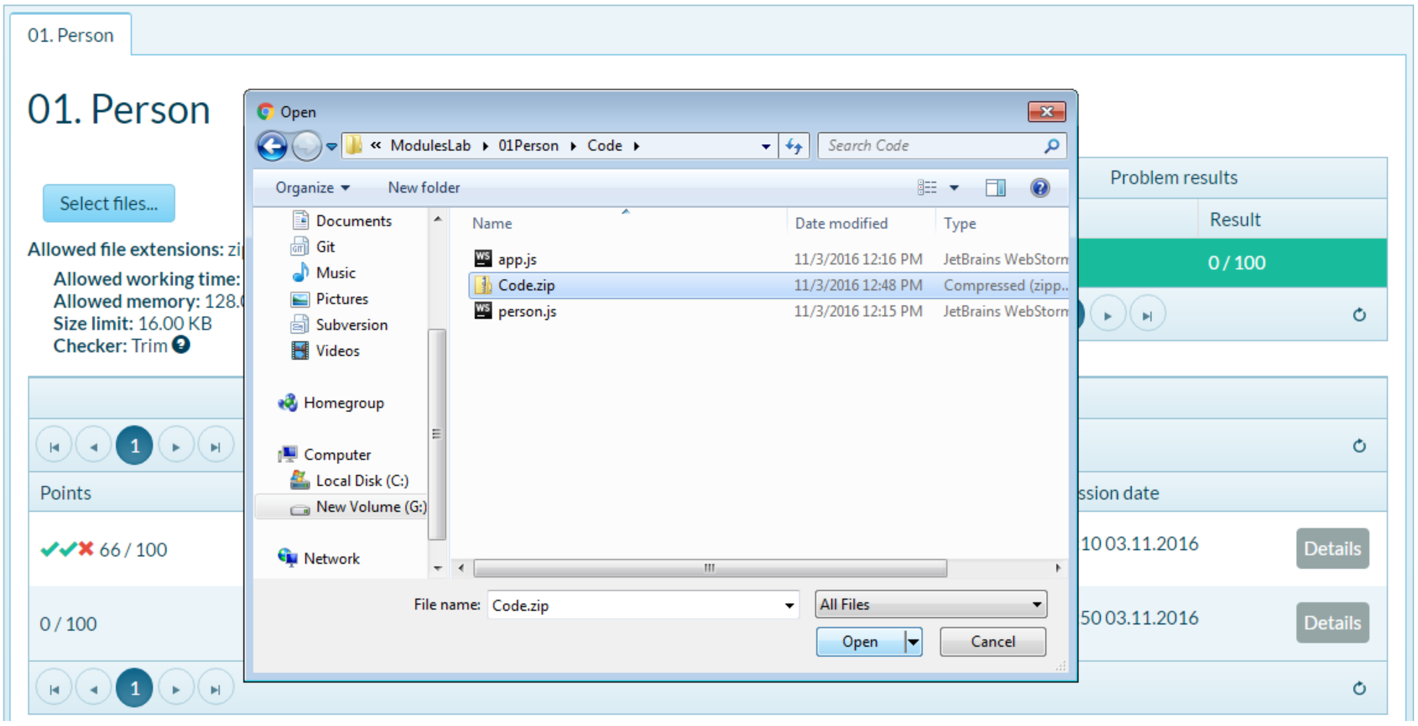
```
let Person = require('./person');

result.Person = Person;
```

If your program is required to return a result (classes, objects, functions etc.) they need to be attached to the **`result`** object in the **`app.js`**.



```
app.js      11/3/2016 12:16 PM   JetBrains WebStorm   1 KB
person.js   11/3/2016 12:15 PM   JetBrains WebStorm   1 KB
```



Allowed file extensions: zi
Allowed working time:
Allowed memory: 128.
Size limit: 16.00 KB
Checker: Trim

ModulesLab ▸ 01Person ▸ Code ▸

| Name | Date modified | Type |
|---|---|---|
| app.js | 11/3/2016 12:16 PM | JetBrains WebStorm |
| Code.zip | 11/3/2016 12:48 PM | Compressed (zipp... |
| person.js | 11/3/2016 12:15 PM | JetBrains WebStorm |

File name: Code.zip
All Files

| Points | | Submission date |
|---|---|---|
| ✓✓✗ 66 / 100 | 10 | 03.11.2016 |
| 0 / 100 | 50 | 03.11.2016 |

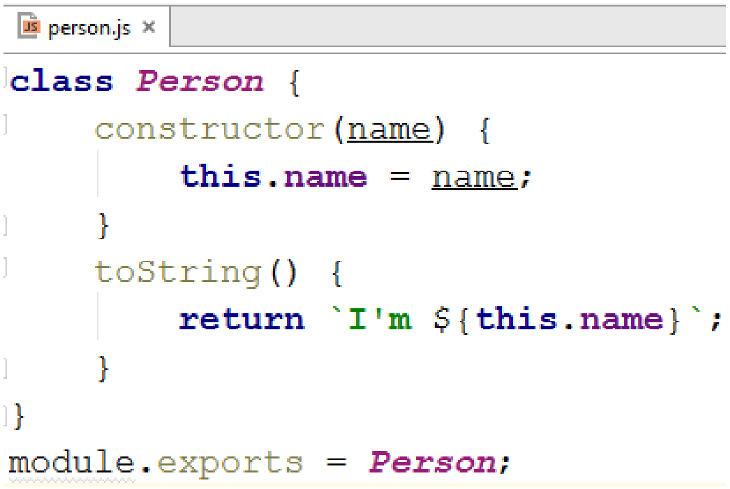
## 1. Person Module

Write a JS module that exports a **class Person**. The Person has a **name** property, that is set trough the constructor and a **`toString()`** method, which returns **"I'm {name}."** as a **string**. Write an **`app.js`** file that loads the **Person** module and attaches it to the **`result`** object provided.

As an exercise, rewrite the task using all three methods for module definition (AMD, CommonJS and ES6 native).

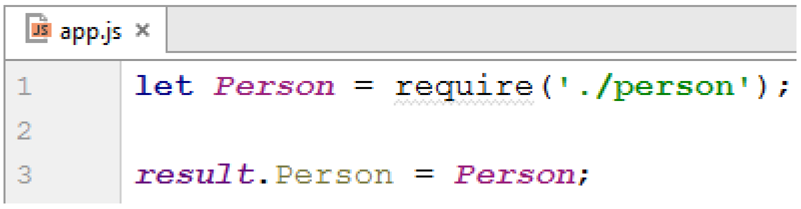
### Screenshots

**`Person.js`**



```
class Person {
    constructor(name) {
        this.name = name;
    }
    toString() {
        return `I'm ${this.name}`;
    }
}
module.exports = Person;
```

**`app.js`**



```
let Person = require('./person');

result.Person = Person;
```

### Bonus

As an exercise, you can rewrite the task using all three methods for module definition (AMD, CommonJS and ES6 native). This part doesn't need to be submitted to the Judge system it is simply for practice.

## 2. Sort Map

You probably wanted to be able to sort a map while going through the Fundamentals course, so we're going to make a simple program to do that. You need to return (attach to result) a method **`mapSort`** which takes in a Map and optionally a **sort function** and **returns** a new **map that is sorted**. In case no **sort function** is supplied the map should be ordered by **comparing its keys**. In order to keep things separate keep the **`mapSort`** function in a different file, you can create a file **`helper-functions.js`** with useful functions to help you when working with JavaScript, **`mapSort`** can be part of these helper functions.

**`mapSort(map, sortFn)`** - returns a new Map that is sorted, in case **`sortFn`** is not supplied the returned map should be sorted by its keys.

| Sample Input | Output |
|---|---|
| `let map = new Map();`<br>`map.set(3,"Pesho");`<br>`map.set(1,"Gosho");`<br>`map.set(7,"Aleks");`<br>`mapSort(map);` | `Map { 1 => 'Gosho', 3 => 'Pesho', 7 => 'Aleks' }` |

### Example

Here's an example template of how your code should look:

**template.js**

```
//Require/Import mapSort

result.mapSort = mapSort;
```

## 3. Load Data

You will be given a file that contains the declaration of an array of object literals. Refactor the file to **export** that array and then write a JS program that **imports** the data and performs queries on it. You are expected to return (attach to result) two functions:

**`sort(property)`** – **return** an array, sorted by the object **property** given as a string argument

**`filter(property, value)`** – **return** an array, containing **only** the objects containing a **property** that matches the given **value**

| Sample Input | Output |
|---|---|
| `sort('shipTo');` | `[ { orderId: 'CRA9938',`<br>`    status: 'delivered',`<br>`    shipTo: 'Baku',`<br>`    latLong: '53.80 33.21' },`<br>`  { orderId: 'XVA7355',`<br>`    status: 'processing',` |

| | |
|---|---|
| | `shipTo: 'Brussels',`<br>`latLong: '54.08 -2.48' },`<br>`{ orderId: 'FMU7330',`<br>`status: 'delivered',`<br>`shipTo: 'Brussels',`<br>`latLong: '63.22 34.78' },`<br>… |
| `filter('status', 'shipped');` | `[ { orderId: 'XVE1808',`<br>`status: 'shipped',`<br>`shipTo: 'Kyiv',`<br>`latLong: '57.24 8.09' },`<br>`{ orderId: 'ABK9047',`<br>`status: 'shipped',`<br>`shipTo: 'Vienna',`<br>`latLong: '51.92 12.10' },`<br>`{ orderId: 'SEL8195',`<br>`status: 'shipped',`<br>`shipTo: 'Kyiv',`<br>`latLong: '60.86 18.32' },`<br>… |

## Example

Here's an example template of how your code should look:

**template.js**

```
//Require the data

function sort(property) {
    //TODO sort and return data
}

function filter(property, value) {
    //TODO filter and return data
}

result.sort = sort;
result.filter = filter;
```

## Bonus

As an exercise, you can rewrite the task using all three methods for module definition (AMD, CommonJS and ES6 native). This part doesn't need to be submitted to the Judge system it is simply for practice.

## 4. Inheritance

You are tasked create a few classes - **Entity**, **Person**, **Student** and **Dog**. Keep each class's in a separate file and import the base class if inheritance is needed. The files should support the following functionality:

- **Entity** - an abstract base class - attempting to create an object of class Entity should throw and Error
    - **name** - string property containing the name of the entity
- **Dog** - a concrete class inheriting Entity
    - **saySomething()** - a method that should return a string in the following format **"{name} barks!".**
- **Person** - a concrete class inheriting Entity
    - **phrase** - string property which is the person's unique phrase
    - **dog** - an object of class Dog
    - **saySomething()** - a method that should return a string in the following format:

**"{name} says: {phrase}{dog.name} barks!"**

- **Student** - a concrete class inheriting Person
    - **id** - a number property containing the student's id
    - **saySomething()** - a method that should return a string in the following format:

**"Id: {id} {name} says: {phrase}{dog.name} barks!"**

### Example

Here's an example template of how your code should look:

| template.js |
|---|
| `//Require/Import each class`<br><br>`result.Entity = Entity;`<br>`result.Person = Person;`<br>`result.Dog = Dog;`<br>`result.Student = Student;` |

Here are some examples for your classes using ES6 syntax:

```
export default class Entity{
    //TODO
}
```

```
import Entity from "../named-entity/entity.js";

export default class Dog extends Entity{
    //TODO
}
```