# Lab: Objects and Classes

Problems with exercise and homework for the ["JS Front-End" Course @ SoftUni.](https://softuni.bg/trainings/3976/js-front-end-february-2023)

## Person Info

Write a function that receives **3 parameters**, sets them to an **object**, and **returns** that object.

The input comes as **3 separate strings** in the following order: **firstName**, **lastName**, **age**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Object Properties** |
| **"Peter",  "Pan", "20"** | **firstName: Peter**  **lastName: Pan**  **age: 20** |
| **"George",  "Smith", "18"** | **firstName: George**  **lastName: Smith**  **age: 18** |

### Hints



## City

Write a function that receives a **single** **parameter** – an **object**, containing **five properties**:

**{ name, area, population, country, postcode }**

Loop through all the **keys** and **print** them with their **values** in format: "**{key} -> {value}**"

See the examples below.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| **{**  **name: "Sofia",**  **area: 492,**  **population: 1238438,**  **country: "Bulgaria",**  **postCode: "1000"**  **}** | **name -> Sofia**  **area -> 492**  **population -> 1238438**  **country -> Bulgaria**  **postCode -> 1000** |
| **{**  **name: "Plovdiv",**  **area: 389,**  **population: 1162358,**  **country: "Bulgaria",**  **postCode: "4000"**  **}** | **name -> Plovdiv**  **area -> 389**  **population -> 1162358**  **country -> Bulgaria**  **postCode -> 4000** |

## City Taxes

*This task is an extension of Problem 1, you may use your solution from that task as a base.*

You will receive a city’s **name** (string), **population** (number), and **treasury** (number)as arguments, which you will need to set as **properties** of an **object** and **return** it. In addition to the input parameters, the object must have a property taxRate with an initial value of **10**, and three **methods** for managing the city:

* collectTaxes() **-** Increase **treasury** by population \* taxRate
* applyGrowth(percentage) **-** Increase population by **given percentage**
* applyRecession(percentage) **-** Decrease treasury by **given percentage**

Round down the values after each calculation.

### Input

Your solution will receive three **valid** parameters. The methods that expect parameters will be tested with valid input.

### Output

Return an **object** as described above. The methods of the object modify the object and don’t return anything.

|  |  |
| --- | --- |
| **Input** | **Output** |
| **const city =**  **cityTaxes('Tortuga',**  **7000,**  **15000);**  **console.log(city);** | {  name: 'Tortuga',  population: 7000,  treasury: 15000,  taxRate: 10,  collectTaxes: [Function: collectTaxes],  applyGrowth: [Function: applyGrowth],  applyRecession: [Function: applyRecession]  } |
| **Testing with code** | |
| **Input** | **Output** |
| **const city =**  **cityTaxes('Tortuga',**  **7000,**  **15000);**  **city.collectTaxes();**  **console.log(city.treasury);**  **city.applyGrowth(5);**  **console.log(city.population);** | 85000  7350 |

## Convert to Object

Write a function that receives a **string** in **JSON format** and converts it to an **object**.

Loop through all the keys and print them with their values in format: "**{key}: {value}**"

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| **'{"name": "George", "age": 40, "town": "Sofia"}'** | **name: George**  **age: 40**  **town: Sofia** |
| **'{"name": "Peter", "age": 35, "town": "Plovdiv"}'** | **name: Peter**  **age: 35**  **town: Plovdiv** |

### Hints

* Use **JSON.parse()** method to parse JSON string to an object



## Convert to JSON

Write a function that receives a **first name**, **last name**, **hair color** and sets them to an **object**.

Convert the **object** to **JSON string** and print it.

Input is provided as **3 single strings** in the order stated above.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| **'George', 'Jones', 'Brown'** | **{"name":"George","lastName":"Jones","hairColor":"Brown"}** |
| **'Peter', 'Smith', 'Blond'** | **{"name":"Peter","lastName":"Smith","hairColor":"Blond"}** |

### Hints

* Use **JSON.stringify()** to parse the object to JSON string



## Phone Book

Write a function that stores information about a **person’s name** and **phone number**. The input is an **array of strings** with space-separated name and number. **Replace duplicate names**. Print the result as shown.

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| **['Tim 0834212554',**  **'Peter 0877547887',**  **'Bill 0896543112',**  **'Tim 0876566344']** | **Tim -> 0876566344**  **Peter -> 0877547887**  **Bill -> 0896543112** |
| **['George 0552554',**  **'Peter 087587',**  **'George 0453112',**  **'Bill 0845344']** | **George -> 0453112**  **Peter -> 087587**  **Bill -> 0845344** |

## Meetings

Write a function that manages meeting appointments. The input comes as an **array of strings**. Each string contains a **weekday** and person’s **name**. For each **successful** meeting, **print a message**. If you receive the **same weekday** twice, the meeting cannot be scheduled so print a **conflicting message**. In the end, print a list of all **successful** meetings.

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| **['Monday Peter',**  **'Wednesday Bill',**  **'Monday Tim',**  **'Friday Tim']** | **Scheduled for Monday**  **Scheduled for Wednesday**  **Conflict on Monday!**  **Scheduled for Friday**  **Monday -> Peter**  **Wednesday -> Bill**  **Friday -> Tim** |
| **['Friday Bob',**  **'Saturday Ted',**  **'Monday Bill',**  **'Monday John',**  **'Wednesday George']** | **Scheduled for Friday**  **Scheduled for Saturday**  **Scheduled for Monday**  **Conflict on Monday!**  **Scheduled for Wednesday**  **Friday -> Bob**  **Saturday -> Ted**  **Monday -> Bill**  **Wednesday -> George** |

## Address Book

Write a function that stores information about a person’s **name** and his **address**. The input comes as an **array of strings**. Each string contains the **name** and the **address** separated by a **colon**. If you receive the same name **twice** just **replace** the address. In the end, print the full list, **sorted alphabetically** by the person’s name.

|  |  |
| --- | --- |
| **Input** | **Output** |
| **['Tim:Doe Crossing',**  **'Bill:Nelson Place',**  **'Peter:Carlyle Ave',**  **'Bill:Ornery Rd']** | **Bill -> Ornery Rd**  **Peter -> Carlyle Ave**  **Tim -> Doe Crossing** |
| **['Bob:Huxley Rd',**  **'John:Milwaukee Crossing',**  **'Peter:Fordem Ave',**  **'Bob:Redwing Ave',**  **'George:Mesta Crossing',**  **'Ted:Gateway Way',**  **'Bill:Gateway Way',**  **'John:Grover Rd',**  **'Peter:Huxley Rd',**  **'Jeff:Gateway Way',**  **'Jeff:Huxley Rd']** | **Bill -> Gateway Way**  **Bob -> Redwing Ave**  **George -> Mesta Crossing**  **Jeff -> Huxley Rd**  **John -> Grover Rd**  **Peter -> Huxley Rd**  **Ted -> Gateway Way** |

## 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** |
| **['Mellow 2', 'Tom 5']** | **Mellow, age 2 says Meow**  **Tom, age 5 says Meow** |
| **['Candy 1', 'Poppy 3', 'Nyx 2']** | **Candy, age 1 says Meow**  **Poppy, age 3 says Meow**  **Nyx, age 2 says Meow** |

### Hints

* 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



## Songs

Define a **class** **Song**, which holds the following information about songs: **typeList**, **name,** and **time**.

You will receive the input as an **array**.

The first element **n** will be the number of songs. Next **n** elements will be the songs data in the following format: **"{typeList}\_{name}\_{time}"**, and the last element will be **typeList** / **"all".**

Print only the **names of the songs**, which have the same **typeList (**obtained as the last parameter**)**. If the value ofthe last element is **"all",** print the names of all the songs.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| **[3,**  **'favourite\_DownTown\_3:14',**  **'favourite\_Kiss\_4:16',**  **'favourite\_Smooth Criminal\_4:01',**  **'favourite']** | **DownTown**  **Kiss**  **Smooth Criminal** |
| **[4,**  **'favourite\_DownTown\_3:14',**  **'listenLater\_Andalouse\_3:24',**  **'favourite\_In To The Night\_3:58',**  **'favourite\_Live It Up\_3:48',**  **'listenLater']** | **Andalouse** |
| **[2,**  **'like\_Replay\_3:15',**  **'ban\_Photoshop\_3:48',**  **'all']** | **Replay**  **Photoshop** |

### Solution:

Create a **Song class** with properties described above



Create a new array, where you will store songs



Iterate over the songs:



