

Lab: Function Context

Problems for the in-class lab for the ["JavaScript Applications" course @ SoftUni](#)

Submit your solutions in the SoftUni judge system at [https://judge.softuni.bg/Contests/Practice/Index/1854 - 0](https://judge.softuni.bg/Contests/Practice/Index/1854-0)

1. Area and Volume Calculator

Write a function which **calculates** the **area** and the **volume** of a figure, which is **defined** by its **coordinates** (**x**, **y**, **z**).

area()

```
function area() {  
    return this.x * this.y;  
};
```

vol()

```
function vol() {  
    return this.x * this.y * this.z;  
};
```

solve()

```
function solve(area, vol, input) {  
    //ToDo....  
}
```

Input

You will receive **3** parameters - the **functions** **area** and **vol** and a **string**, which contains the figures' coordinates.

For more information check the examples.

Output

The output should be **returned** as an **array of objects**. Each object has **two properties**: the figure's **area** and **volume**.

```
[  
  { area: ${area1}, volume: ${volume1} },  
  { area: ${area2}, volume: ${volume2} },  
  . . .  
]
```

Note:

Submit only the solve function.

Examples

Sample Input	Output
<pre>area, vol, '[{"x":"1","y":"2","z":"10"}, {"x":"7","y":"7","z":"10"}, {"x":"5","y":"2","z":"10"}]'</pre>	<pre>[{ area: 2, volume: 20 }, { area: 49, volume: 490 }, { area: 10, volume: 100 }]</pre>
<pre>area, vol, '[{"x":"10","y":"-22","z":"10"}, {"x":"47","y":"7","z":"-5"}, {"x":"55","y":"8","z":"0"}, {"x":"100","y":"100","z":"100"}, {"x":"55","y":"80","z":"250"}]'</pre>	<pre>[{ area: 220, volume: 2200 }, { area: 329, volume: 1645 }, { area: 440, volume: 0 }, { area: 10000, volume: 1000000 }, { area: 4400, volume: 1100000 }]</pre>

2. 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 **accessibles**, 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}"`

Note: For more information check the examples below.

Examples

Sample Input
<pre>let person = new Person("Peter", "Ivanov"); console.log(person.fullName);<i>//Peter Ivanov</i> person.firstName = "George"; console.log(person.fullName);<i>//George Ivanov</i> person.lastName = "Peterson"; console.log(person.fullName);<i>//George Peterson</i> person.fullName = "Nikola Tesla"; console.log(person.firstName)<i>//Nikola</i> console.log(person.lastName)<i>//Tesla</i></pre>
<pre>let person = new Person("Albert", "Simpson"); console.log(person.fullName);<i>//Albert Simpson</i></pre>

```
person.firstName = "Simon";  
console.log(person.fullName); //Simon Simpson  
person.fullName = "Peter";  
console.log(person.firstName) // Simon  
console.log(person.lastName) // Simpson
```

3. ArrayMap

Write a function that takes **2 parameters** (**array** and a **function**) that uses **.reduce()** to implement a simple version of **.map()**.

Input

You will receive **2 parameters** - an **array**, and a **function**.

Output

The output should be **returned** as a **new array** (changed according to the given function).

For more information check the examples below.

Examples

Sample exectuion
<pre>let nums = [1,2,3,4,5]; console.log(arrayMap(nums,(item)=> item * 2)); // [2, 4, 6, 8, 10]</pre>
<pre>let letters = ["a","b","c"]; console.log(arrayMap(letters,(l)=>l.toLocaleUpperCase())) // ['A', 'B', 'C']</pre>

4. Dropdown Menu

Use the Given Skeleton to Solve This Problem.

Note: You Have NO Permission to Change Directly the Given HTML (Index.html File).

```
▼<div class="container">
  <button id="dropdown">
    Choose your style
  </button>
  ▼<ul id="dropdown-ul">
    <li class="deep">rgb(255, 143, 143)</li>
    <li class="deep1">rgb(250, 215, 151)</li>
    <li class="deep2">rgb(251, 251, 167)</li>
    <li class="deep3">rgb(228, 255, 173)</li>
    <li class="deep4">rgb(174, 174, 251)</li>
  </ul>
</div>
<div id="box">Box</div>
```

Choose your style

Box

Your Task

Write the missing JavaScript code to make the **Dropdown Menu** application work as expected.

When you **click** on the [Choose your style] button, the elements of the menu should become visible.

```
▼<div class="container">
  <button id="dropdown">
    Choose your style
  </button>
  ▼<ul id="dropdown-ul" style="display: block;">
    <li class="deep">rgb(255, 143, 143)</li>
    <li class="deep1">rgb(250, 215, 151)</li>
    <li class="deep2">rgb(251, 251, 167)</li>
    <li class="deep3">rgb(228, 255, 173)</li>
    <li class="deep4">rgb(174, 174, 251)</li>
  </ul>
</div>
<div id="box">Box</div>
```

Choose your style

rgb(255, 143, 143)

rgb(250, 215, 151)

rgb(251, 251, 167)

rgb(228, 255, 173)

rgb(174, 174, 251)

Box

When you click on one of the items the background color of the box below should be changed to the RGB, which is displayed in the list item

```
<div class="container">
  <button id="dropdown">Choose your style</button>
  <ul id="dropdown-ul" style="display: block;">
    <li class="deep"></li>
    <li class="deep1"></li>
    <li class="deep2"></li>
    <li class="deep3"></li>
    <li class="deep4"></li>
  </ul>
</div>
<div id="box" style="background-color: rgb(251, 251, 167); color: black;">Box</div>
<!--Code injected by live-server-->
```

Choose your style

rgb(255, 143, 143)

rgb(250, 215, 151)

rgb(251, 251, 167)

rgb(228, 255, 173)

rgb(174, 174, 251)

Box

Choose your style

rgb(255, 143, 143)

rgb(250, 215, 151)

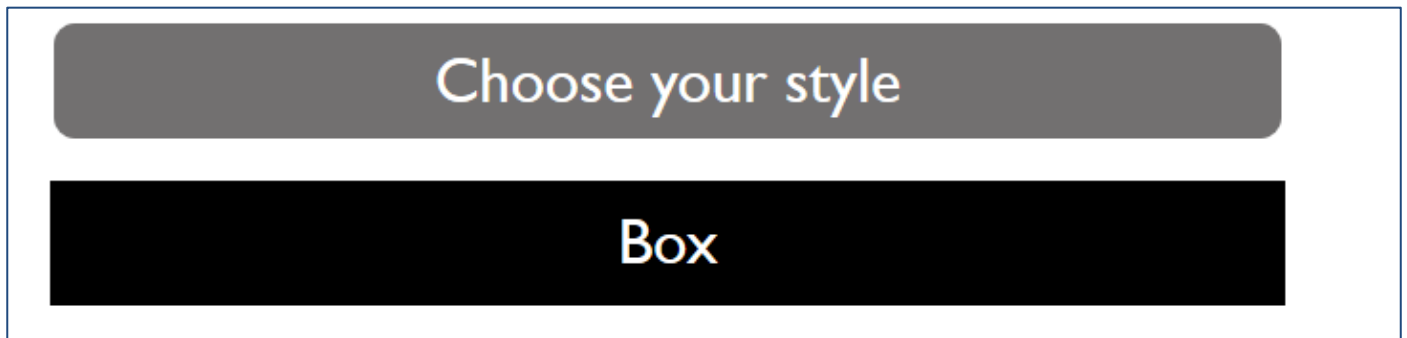
rgb(251, 251, 167)

rgb(228, 255, 173)

rgb(174, 174, 251)

Box

When the [Choose your style] button is clicked again, you should hide the list items, and the box should be returned to its initial state.



5. Spy

Write a function that takes 2 parameters **target**(an object) and **method**(a string) and tracks **how many times** the method of an object is **called**.

Input

- **target**: an **object** containing the **method**
- **method**: a **string** with the **name of the method** on target to spy on

Output

The output should be **returned** as an **object** with property **count**, which holds how many times the provided method is invoked.

Examples

Sample execution
<pre>let obj = { method:()=> "invoked" } let spy = Spy(obj, "method"); obj.method(); obj.method(); obj.method(); console.log(spy) // { count: 3 }</pre>
<pre>let spy = Spy(console, "log"); console.log(spy); // { count: 1 } console.log(spy); // { count: 2 } console.log(spy); // { count: 3 }</pre>

Hints

Check the code below.

```
function Spy(target, method) {
  let originalFunction = target[method]

  // use an object so we can pass by reference, not value
  // i.e. we can return result, but update count from this scope
  let result = {
    count: 0
  }

  // replace method with spy method
  target[method] = function () {
    result.count++ // track function was called
    return originalFunction.apply(this, arguments) // invoke original function
  }

  return result
}
```