

```
> function show(callBackFn){  
    for(var i = 1; i<=10;i++){  
        callBackFn(i);  
    }  
< undefined  
> function cube(num){  
    console.log(num**3);  
}  
< undefined  
> show(cube);
```

+ Show(cube())
Show(cube)

```
> function calc(){  
    // nested function  
    var add = function(x,y){  
        return x + y;  
    }  
    return add;  
}
```

```
> var f = calc();  
< undefined  
  
> f;  
< f (x,y) {  
    return x + y;  
}  
  
> typeof f;  
< 'function'  
  
> f(10,20);  
< 30
```

```
> // Imperative Statements
< undefined
> var arr = [10,20,30,40,50,1,2,40,100,40];
< undefined
> var search = 40;
< undefined
> var
  for(var i = 0; i<arr.length; i++){
    if(arr[i] == search){
      count++;
    }
  }
> var arr2 = [];
  for(var i = 0; i<arr.length; i++){
    if(arr[i] == search){
      arr2.push(arr[i]);
    }
  }
< 3
> arr2;
< ▶ (3) [40, 40, 40]
```

```
> var arr2 = [];
  for(var i = 0; i<arr.length; i++){
    if(arr[i] == search){
      arr2.push(arr[i]);
    }
  }
< 3

> arr2;
< ▶ (3) [40, 40, 40] (Callback)

> arr2.length;
< 3

> arr.filter((e)=>e==40); (Declarative)
< (3) [40, 40, 40]
```

Clone

```
> var arr = [10,20,30,40,10];
< undefined
> var e = [...arr];
< undefined
> e;
< ◀ (5) [10, 20, 30, 40, 10]
> arr === e;
< false
<
> var f = arr.filter(w=>true);
< undefined
> f;
< ◀ (5) [10, 20, 30, 40, 10]
> f === arr;
< false
```

```
> // filter vs map
< undefined
```

```
> arr;
```

```
< ▶ (5) [10, 20, 30, 40, 10]
```

```
> arr;
```

```
< ▶ (5) [10, 20, 30, 40, 10]
```

```
> arr.map(e=>e + e * 0.18);|
```

```
< (5) [11.8, 23.6, 35.4, 47.2, 11.8]
```

```
> var h = arr.map(e=>e);
```

```
< undefined
```

```
> h;
```

```
< ▶ (5) [10, 20, 30, 40, 10]
```

```
> h === arr;
```

```
< false
```

A screenshot of a browser's developer tools Console tab. The console shows the following code execution:

```
< ◀ (4) ['1,000', '20,000', '4,00,000.2', '5,555']  
> var formattedPrices = price.map(p=>Rs p.toLocaleString('hi'));  
< undefined  
> price;  
< ◀ (4) [1000, 20000, 400000.2, 555555] price
```

Handwritten annotations include:

- An oval labeled "map" with arrows pointing to the `map` keyword and the parameter `p` in the arrow function.
- A circle labeled "price" with an arrow pointing to the variable declaration.
- A bracket under the array elements with an arrow pointing to the value "5,555" and the label "5 times 1000".
- A bracket under the array elements with an arrow pointing to the value "4,00,000.2" and the label "5 times 10000".
- A bracket under the array elements with an arrow pointing to the value "1,000" and the label "5 times 100000".
- A bracket under the array elements with an arrow pointing to the value "20,000" and the label "5 times 1000000".

A screenshot of a browser's developer tools Console tab. The console shows the following code execution:

```
> var arr = [10,20,30,40,10];  
< undefined  
> arr.forEach(e=>console.log(e));  
10  
20  
30  
40  
10  
< undefined
```

Handwritten annotations include:

- A bracket under the array elements with an arrow pointing to the value "10" and the label "function".
- A bracket under the array elements with an arrow pointing to the value "10" and the label "Callback".
- Five separate arrows pointing from the values 10, 20, 30, 40, and 10 to the right, labeled "VM162:1" below each one.

```
> arr.forEach((element, index, a)=>console.log(element, index, a));  
10 0 ▶ (5) [10, 20, 30, 40, 10] VM277:  
20 1 ▶ (5) [10, 20, 30, 40, 10] VM277:  
30 2 ▶ (5) [10, 20, 30, 40, 10] VM277:  
40 3 ▶ (5) [10, 20, 30, 40, 10] VM277:  
10 4 ▶ (5) [10, 20, 30, 40, 10] VM277:
```

```
> arr.find(e=>e==10)?"Found":"Not found";  
< 'Found'  
> arr.findIndex(e=>e==10);  
< 0  
> arr;  
< ▶ (5) [10, 20, 30, 40, 10]  
> arr.findIndex(e=>e==1000);  
< -1
```

```
> arr.filter(e=>e>20);
```

```
< ◀ (2) [30, 40]
```

```
> arr.reduce((result, e)=>{  
    if(e>20){  
        result.push(e);  
    }  
    return result;  
, []);
```

```
< ◀ (2) [30, 40]
```

```
> arr.reduce((result, e, i)=>{  
    if(e>20){  
        result.push({element: e, index : i});  
    }  
    return result;  
, []);
```