**Example 1**

var materials = [

'Hydrogen',

'Helium',

'Lithium',

'Beryllium'

];

**Type 1:**

var res2 = materials.map(function(material){

return material.length;

})

console.log(res2)

**Type 2:**

var res1 = materials.map((material) => {

return material.length;

})

console.log(res1);

**Type 3:**

var res = materials.map(material => material.length)

console.log(res);

In the above example ,

-> Actual code will be like **TYPE-1 with function**

**->** we can reduce the code as **TYPE-2 by removing the function from Type-1,**if we want to reduce the code more, then

**-> OneLine**code can be writtenlike**Type-3 by removing the return statement from Type-2**

**Example 2**

**wrong:** console.log(typeof () => {});

**correct:** console.log(typeof **(**() => {}**)**);

In the above example Usage of Typeof() method is declared

**Example 3**

const dragonEvents = [

{type:'attack',value:12, target:'player-dorkman'},

{type:'yawn',value:40},

{type:'attack',value:23, target:'player-1'},

{type:'attack',value:12, target:'player-dorkman'},

]

**ddType 1:**

const newdata = dragonEvents

.filter(function(event) {

return event.type === 'attack'

})

.filter(function(event) {

return event.target === 'player-dorkman'

})

.map(function(event) {

return event.value

})

.reduce(function(prev,value) {

return (prev || 0) + value

})

console.log('newdata\n',newdata);

**Type-2**

const newdata = dragonEvents

.filter( (event) => {

return event.type === 'attack'

})

.filter((event) => {

return event.target === 'player-dorkman'

})

.map( (event) => {

return event.value

})

.reduce((prev, value) => {

return (prev || 0)+value

})

console.log('newdata\n',newdata);

**Type-3**

const twoval = (prev, value) => (prev || 0)+value

const eventval = event => event.value

const eventval1 = event => event.target === 'player-dorkman'

const eventval3 = event => event.type === 'attack'

const newdata = dragonEvents

.filter(eventval3)

.filter(eventval1)

.map(eventval)

.reduce(twoval)

console.log('newdata\n',newdata);

In the above example ,

-> actual code will be like **TYPE-1 with functions**

**->** we can reduce the code as **TYPE-2 by removing the functions from Type-1,**if we want to reduce the code more, then

**-> OneLine**code can be writtenlike**Type-3 by removing the return statements from Type-2**

**Example 4**

function Person() {

this.age = 0;

setInterval(() => {

console.log(this.age);

this.age++;

}, 1000);

}

var p = new Person();

Omitting parentheses around single parameters

->

[1,2,3].map(x => 2 \* x)

output: [ 2, 4, 6

]

-> [[1,2],

[3,4]].map(([a,b]) => a + b)

output: [ 3, 7 ]

-> [1, undefined,

3].map((x='yes') => x)

output: [ 1,

'yes', 3 ]

**Some more examples on**

**Arrows**

->

const x= 122

const

f = **y**

=> (x % 2) === 0 ? x : 0;

console.log(f())

OUTPUT

: 122

-> const x= 122

const f = **x**=> (x % 2) === 0 ? x : 0;

console.log(f())

OUTPUT : 0

-> const = 123

const f = x => (x % 2) === 0 ? x : 0;

console.log(f())

OUTPUT : 0

->

const x= 122

const

f = **y**

=> ((x % 2) === 0 ? x : 0);

console.log(f())

OUTPUT

: 122

-> const x= 122

const f = **x** => ((x % 2) === 0 ? x : 0);

console.log(f())

OUTPUT : 122

-> const = 123

const f = y => ((x % 2) === 0 ? x : 0);

console.log(f())

OUTPUT : 0  
-> const f1 = x => ({ bar: 123 });

console.log(f1())

OUTPUT : Object { "bar": 123 }

**Examples on LET and VAR**

-> **USING LET**

const key = 'abc123';

**let** points = 50;

**let** winner = false;

console.log(winner)

if(points > 40) {

**let** winner = true

console.log(winner)

}

console.log(winner)

**OUTPUT:**

**false**

**true**

**false**

-> **USING VAR**

const key = 'abc123';

**var** points = 50;

**var** winner = false;

console.log(winner)

if(points > 40) {

**var** winner = true

console.log(winner)

}

console.log(winner)

**OUTPUT:**

**false**

**true**

**true**

**Example 7**

**Using const keyword**

-> **const** key = 'abc123';

let points = 50;

let winner = false;

key = 'abc1234';

console.log(key)

**OUTPUT**

**ERROR**

**Using LET keyword**

-> **let** key = 'abc123';

let points = 50;

let winner = false;

key = 'abc1234';

console.log(key)

**OUTPUT**

**"abc1234"**