Ex

```
let items = num
        .filter(value => value >=0)
        .map(~~number~~number => {value: number});
```

// In the above code we replaced filtered
// variable with its original code.

* Functions

A block of code that fulfills a specific task is called a function.

Creation Syntax

```
function funName() {
          statement
}
```

Why we need functions?

→ Better readability

→ code reuse

→ To avoid bulky code

→ To reduce bugs

* Function Declaration

```
function run(parameters) {
        console.log('running');
}
```

* Function calling / invoking

```
run();
```

* Hoisting Concept

Q. If we call the function before the actual function is written, will it run properly?

⇒ Yes, because in JavaScript all the functions are sent at the top of the file and this process is known as Hoisting and it's done autometically by js engine.

* Types of function assignment

i) Name function assignment

ii) Anonymous function assignment

Named function Assignment

```
let stand = function walk() {
        console.log('walk');
}
```

Anonymous Function Assignment

```
let stand2 = function() {
        console.log('walk');
}
```

Calling the function

```
stand()
```

// If we call the walk() after the assignment it will not be called, we can only call it using stand().

Can we call the function before its declaration?
No, because hoisting only works with function initialization not with function assignment.

JS is Dynamic Language

```
let n = 1;
n = 'a';
```

No error will come in the above code.

```
function sum(a,b) {
    return a+b;
}
```

console.log(sum(1,2)) --> 3

console.log(sum(1)) --> If we pass only one value the the output will be NaN (Not a Number) because by default value 2 will be not defined.

console.log(sum()) --> NaN

console.log(sum(1,2,3,4,5)) --> 3 because 1 -> a and 2 -> b and rest of the numbers will go to arguements.

Arguements are special ~~of~~ objects.

* REST operator (...)

It is also known as spread operator, it is used to concatinate & copy the arrays.

If in a function we have multiple parameters the we can handle all the parameters with the help of the rest operator.

```
function sum (num, value, ...args) {
        console.log (args)
}

sum (1,2,3,4,5,6,7,8)
```

⇒ By using the rest parameter we can ~~ol~~ only assign the values before args not after it.

⇒ We can store the varying parameters / data in the array form using the rest operator.

* Default Parameters

When a user does not pass any value to the function then the function takes the value by itself.

⇒ Below r=5 is default parameter

So, if you pass a value to 'r' in the function calling then it will take the passed value otherwise it will take the default value.

```
function interest (p, r=5, y=10) {
        return p*r*y/100 ;
}

console.log(interest(1000)) ;
```

⇒ If 'r' is a default parameter then all the parameter to the right side of it must be also default. (It's a rule)

* Getter Setter

getter → to access the properties

setter → to change or mutate properties

```
let person = {
    fName : 'Rishabh',
    lName : 'Kushwaha',
getter  get fullName() {
            return `${person.fName} ${person.lName}`
        },

setter  set fullName(value) {
            let parts = value.split(' ');
            this.fName = parts[0];
            this.lName = parts[1];
        }
};

console.log(person.fullName);
→ Rishabh Kushwaha

person.fullName = 'Love Babbar';

console.log(person.fullName);
→ Love Babbar
```

* Try and Catch Block

=> The try statement defines a code block to run (to try).

=> The catch statement defines a code block to handle any error.

=> The finally statement defines a code block to run regardless of the result.

=> The throw statement defines a custom error.

```
try {
    Block of code to try
}
catch (err) {
    Block of code to handle errors
}
finally {
    Block of code to run regardless of the try/
    catch result
}
```

* Scope

Scope determines the accessibility (visibility) of variable.

JavaScript has 3 types of scopes:

-> Block scope

-> Function scope

-> Global scope

Ex

```
{
    var n = 2;
    let y = 2;
}

// X can be used here
// y cannot be used here
```

Variable declared with var keyword cannot have block scope.

* Reducing an Array

=> The reduce() method executes a reducer function for an array.

=> The reduce() method returns a single value: the function's accumulated result.

=> The reduce() method does not change the original array.

```
let arr = [1,2,3,4,5];

let sum = 0;

for (let value of arr) {
        sum = sum + value;
}
console.log (sum);
-> 15
```

Now by using reduce() method with => function

```
let sum1 = arr.reduce((accumulator, currenValue) =>

accumulator + currentValue, 0);
```

/*-> accumulator is like the sum variable that stores the value.

-> currentValue is like the loop

-> 0 is initialization of the value (if 0 is not set then the accumulator will start from the first value i.e. 1)

```
console.log(sum1);
-> 15
```