```
// ============== let and const ===========

/*
 when the variable only remanings constant then create => const

        var
       /    \
    let     const

    1) variable values => let
    2) scope => let
    3) constant values => const
*/


// ========== Arrow Functions =========
/*
    Arrow is spacial way to create function

    function myFunc(){
        .....
    }

    const myFunc = () =>{
        ........
    }

*/
    function myFunc(name){

        console.log(name); // "Mehedi"

    }
    myFunc("Mehedi");

    // Arrow function
    const myArrow = (name) =>{

            console.log(name); // "Mehedi"

    }

    myArrow("Mehedi");

    // short way to create function

    const shortV = ParaMeterSingle => ParaMeterSingle * 3;

    // call the function and get value return value
    console.log(shortV(5));  // 15


// =============== Exports & imports (Modules) ==========

 //     1) exports means send
 //     2) imports means recieve

 // let's say
 /*
   one file person.js

    const person = {
        name: "Mehedi";
    }
    export default person;
```

```
66
67      // Import default and ONLY export of the file Name in the
68      //reciveing the file is up to you
69   */
70  /*
71      second file utility.js
72
73      export const clean = ()=> {...};
74      export const baseData = 10;
75  */
76  /*
77      app.js
78
79      import person from './person.js';
80      import prs from './person.js';
81
82      import {baseData} from './utility.js';
83      import {clean} from './utility.json';
84  */
85
86  /*
87      default export => 1) import person from './person';  // file name or rename file
88                        2) import prs form './person.js'; // rename of the person file and recerive
89
90      Named export or variable => 1) import {smth} form '.utility.js';  //excat the vaiable name given
91                                  2) import {smth as Smth} from './utility.js'; // rename the variable name
92                                  3) import *as bundle from '.utility.js'; // All varibale receive and use like -- bundle.clean or bundle.baseData
93  */
94
95
96  // ============ Classes ===============
97
98  /*
99      class Person {
100         name = "Mehedi"; // property
101         call = ()=> {....}; // Method
102     }
103     const myPerson = new Person();
104     myPerson.call();
105     console.log(myPerson.name);
106 */
107
108
109
110 class Human {
111     constructor(){
112         this.gender = "Male";
113     }
114     printGender(){
115         console.log(this.gender);
116     }
117 }
118
119 // class can be inharient
120
121 class Person extends Human{
122     constructor(){
123         // if we use Constractor inside the child class then we have to use
124         super();
125         this.name = "Mehedi";
126         // we can use the parent class property insid the child class if we inharite
127         this.gender = "M";
128     }
129     printName(){
130         console.log(this.name);
131     }
```

```
}
const person = new Person();
person.printName();
person.printGender(); // After changing value it will show => "M"


// ============ Classes, Properties & Mehtods ========


/*
    Properies are like "variables" attached to classes / object
    ES6
    constractor(){
        this.myProperty = 'value';
    }
    ES7
    myProperty = "value";

*/
/*
    Methods are like "function" attached to classes / Object
    ES6
    myMethod(){
        ....
    }
    ES7
    myMethod = ()=> {
        .......
    }
*/

class parent {
    gender = "Male";

    printGender = ()=>{
        console.log(this.gender);
    }
}
class child extends parent {
    name = "Hasan";
    gender = 'MM'; // inharite parent and change value

    printMyName = ()=>{
        console.log(this.name);
    }
}

// create object of child class
    const child1 = new child();
    child1.printMyName(); // "Hasan";
    child1.printGender(); // MM


// ========== Spread & rest Operators =========
  /*                  ...
                  three dot

      Spread => used to split up array elements OR object properties
          const newArray = [...OldArray, 1, 2, 3]; // copy of old array and create new array with 1,2,3 value as well
          const newObject = {...oldObject, newProp: 5}; // copy old object one or more property can be add

    Rest => Used to merge a list of function arguments into an array
          function sortArgs(...args){
              return args.sort();
          }
          // this function take one or more number of arguments
```

```
197         // this function take one or more number of arguments
198
199 */
200     const oldArray = [1, 2, 3];
201     const newNumber = [...oldArray, 4, 5];
202     console.log(newNumber); // (5) [1, 2, 3, 4, 5]
203
204
205     const oldObject = {
206
207         Name: "Mehedi",
208         Address: "Dhaka",
209     };
210     const newObject = {
211         ...oldObject,
212         age: 30,
213     }
214     console.log(newObject); // {Name: "Mehedi", Address: "Dhaka", age: 30}
215
216     const filter = (...arr) =>{
217         return arr.filter((el) => el === 1);
218     }
219
220     console.log(filter(2, 1, 2, 1, 1 , 3, 4)); // 3) [1, 1, 1]
221
222 // ============= Destructuring ================
223 /*
224     Easily extract array elements or object properties and store them in variables.
225     Array
226     [a, b] = ["Hello", "Mehedi"];
227     console.log(a); // Hello
228     console.log(b); // Mehedi
229
230     Object
231     {name} = {name: "Mehedi", age: 28};
232     console.log(name); // "Mehedi";
233     console.log(age); // undefined
234
235 */
236     const Arr = [1, 2, 3, 4];
237     [a, b] = Arr;
238     console.log(a); // 1
239     console.log(b); //2
240     [a, ,b] = Arr;
241     console.log(b); // 3
242
243 // ========= Reference and Primitive ==========
244
245     const num  = 1; // primitive
246     const num2 = num;
247
248     // but if we use Reference then it will
249     //not copy but it will indicate same ponter
250     const person2 = {
251         name: "Mehedi",
252     };
253     const secondPerson = person2;
254     console.log(secondPerson); // {name: "Mehedi"};
255     person2.name = "Hasan";
256     console.log(secondPerson); // {name: "Hasan"};
257
258     // But this is hoy we can copy object
259     const thirdPerson = {
260         ...person, // copy object pass by value or property
261         age: 30,
262     }
```

```
263
264 // ========== Refershing Array ========
265
266     const num5 = [1, 2, 3];
267     const doubleNumArray = num5.map( (e)=>{
268         return e * 3;
269     });
270
271     console.log(num5); // (3) [1, 2, 3]
272     console.log(doubleNumArray); //(3) [3, 6, 9]
```