es6 new syntax

arrow functions

```
"use strict"


//---------------------------------------
// Function Expression
//---------------------------------------


// let getPrice = function () {
//     return 100 + 200
// }

// - or -

// let getPrice = () => {
//     return 100 + 200
// }


// --------------------------------------------------


// let getPrice = (count) => {
//     return count * (100 + 200)
// }

//- or -


// let getPrice = count => {
//     return count * (100 + 200)
// }

// --------------------------------------------------

// let getPrice = (count, tax) => {
//     return count * (100 + 200) + tax
// }

// -or-


// let getPrice = (count, tax) => count * (100 + 200) + tax


// --------------------------------------------------


// let getPrice = (count, tax) => {
//     let cost = count * (100 + 200)
//     let total = cost + tax
//     return total
// }

// --------------------------------------------------
// why/where we need arrow function ?
// --------------------------------------------------
// #1 for compact function-argument :
// --------------------------------------------------

let numbers = [1, 3, 5, 7, 9, 2, 4, 6, 8, 10]
numbers.sort()
numbers.sort(function (x, y) { return x - y })
// -or-
numbers.sort((x, y) => { return x - y })
// -or-
numbers.sort((x, y) => x - y)

/**
 *
 *  sorting
 *  -------
 *
 *      step-1 : compare
 *      step-2 : swap
 *
 *
 *
 */


// -------------------------------------------------
// Quiz:
//---------------------------------------------------

// let tnr = {
//     name: 'Nag',
//     doTeach: function () {
//         console.log(`${this.name} teaching...`)
//         let self = this
//         let askQues = function (q) {
//             console.log(`${self.name} answering ${q}`)
//         }
//         console.log(`teaching ends`)
//         return askQues
//     }
// }
// // today
// let askQues = tnr.doTeach()
// askQues("Q1")
// askQues("Q2")

// // tomorrow
// let tempTnr = {
//     name: 'kishore'
// }
// askQues = tnr.doTeach.call(tempTnr)
// askQues("Q3")


// --------------------------------------------------
// #2 to capture 'this'
// --------------------------------------------------

let tnr = {
    name: 'Nag',
    doTeach: function () {
        console.log(`${this.name} teaching...`)
        // let askQues = function (q) {
        //     console.log(`${this.name} answering ${q}`)
        // }
        // -or-
        let askQues = (q) => {
            console.log(`${this.name} answering ${q}`)
        }
        console.log(`teaching ends`)
        return askQues
    }
}
// // today
// let askQues = tnr.doTeach()
// askQues("Q1")
// askQues("Q2")

// // // tomorrow
// let tempTnr = {
//     name: 'kishore'
// }
// askQues = tnr.doTeach.call(tempTnr)
// askQues("Q3")

//----------------------------------------------------------------------

// // in global-scope

// // console.log(this)
// let normalFunc = function () {
//     console.log(this)
// }
// // normalFunc()

// let arrFunc = () => {
//     console.log(this)
// }
// // arrFunc()


// let person1 = {
//     name: "Nag"
// }
// let person2 = {
//     name: "Indu"
// }
// person1.normalFunc = normalFunc // static function binding
// person1.normalFunc()
// person1.arrFunc = arrFunc // static function binding
// person1.arrFunc()

// normalFunc.call(person2)
// arrFunc.call(person2)


//----------------------------------------------------------------------


function teach() {
    console.log(this.name + " teaching")
    // let askQues = function () {
    //     console.log(this.name + " answering ques")
    // }
    // -or-
    let askQues = () => {
        console.log(this.name + " answering ques")
    }
    return askQues
}

let nag_tnr = { name: 'Nag' }
let nag_askQues = teach.call(nag_tnr)
// nag_askQues.call(nag_tnr)
// or
nag_askQues()

let ki_tnr = { name: 'Kishore' }
```

```
nag_askQues.call(ki_tnr) // it never happened, becoz arrow function cannot bind with any other object

// ----------------------------------------------------------
```

destructuring

```
// object de-structuring


let person = {
    name: 'Nag',
    age: 36
}

// manual de-structuring

// let myName = person.name
// let myAge = person.age


// or


// let { name: myName, age: myAge } = person


//------------------------------------------


// let name = person.name
// let age = person.age

// or

// let { name: name, age: age } = person
// or
// let { name, age } = person



// array destructuring


let arr = [1, 2, 3, 4, 5, 6, 7, [8, 9]]

// let n1 = arr[0]
// let n2 = arr[1]
// let n3 = arr[2]
// let n4 = arr[3]

// or

let [n1, n2, n3, n4, n5 = 50, , n7, [n8, n9]] = arr
```

---

spread operator

```
//----------------------------------------------------------
// spread operator
//----------------------------------------------------------

// function func(a, b, c, d) {
//     console.log(a);
//     console.log(b);
//     console.log(c);
//     console.log(d);
// }

// func(10, 20, 30)

// let numbers = [10, 20, 30, 40]

// func(numbers)

// func(numbers[0], numbers[1], numbers[1])

// func(...numbers) // spread

//----------------------------------------------------------

let arr1 = [1, 2, 3]
let arr2 = [7, 8, 9]
let str = "cts"
let arr3 = [...arr1, 4, 5, 6, ...arr2, ...str]

//----------------------------------------------------------

let o1 = { x: 10 }
let o2 = { y: 20, z: 30 }
let o3 = { ...o1, k: 1, ...o2 }

//----------------------------------------------------------


let arr = [1, 20, 40, 70, 4]
let maxNum = Math.max(...arr)
```

iterables

```
//------------------------------------
// iterable objects
//------------------------------------

let menu = [
    "idly",
    "vada",
    "poori"
]

// for (let i = 0; i < menu.length; i++) {
//     let menuItem = menu[i]
//     console.log(menuItem)
// }

// for (let item of menu) {
//     console.log(item)
// }
// let [m1, m2, m3] = menu

// let newMenu = [...menu, 'pongal']


//-----------------------------------------------
// custom iterable object
//-----------------------------------------------


let idGenerator = {
    [Symbol.iterator]: function () {
        let id = 0;
        return {
            next: function () {
                console.log("next()")
                id++
                let value = id <= 10 ? id : undefined
                let done = id <= 10 ? false : true
                return { value, done }
            }
        }
    }
}

// for (let id of idGenerator) {
//     console.log(id);
// }

// let [id1, id2] = idGenerator

let myIds=[...idGenerator]
```

obj literal enhancements

```
//--------------------------------
// obj literal enhancements
//--------------------------------

//--------------------------------
// ES5
//--------------------------------
let name = "Nag"
let age = 36

let person = {
    name: name,
    age: age,
    sayName: function () {
        //..
    },
    3: 'three',
    'home-address': 'chennai'
}

//--------------------------------
// ES6
//--------------------------------

let addressType = "office"  // office | vacation
let person2 = {
    name,
    age,
    sayName() {
        //..
    },
    [1 + 2]: 'three',
    [addressType + "-address"]: 'chennai-india',
    'say Hi'() {
        console.log("hi")
    }
}

person2['say Hi']()
```