**JAVASCRIPT LANJUTAN | 8.2 Rest Parameter**

Seperti yg kalian tau kita udah bahas spread operator yg notasi titik 3 seperti ini ( **…** ), nah kali ini yg akan kita bahas mengenai rest parameter notasi nya masih sama menggunkan titik 3 tadi nah tapi untuk ini kita gunakan untuk hal yg berbeda.

Sebetulnya kita udah pernah coba ini saat kita menggunakan Destructuring. Definisi nya : *“Merepsentasikan argument pada function dengan jumlah yang tidak terbatas menjadi sebuah array.”* Kalian tau kan kalo didalam function kita bisa argument yg argument nya kita bisa tangkap kedalam varibale nah tapi dengan menggunakan rest parameter ini kita bisa tangkap banyak argument sampai tidak terbatas untuk menajdi sebuah array.

//LETS CODE!!

Misalnya kita punya function namanya myFunc, nah kita panggil myFunc nya yg berisi 5 buah parameter angka begini (1, 2, 3, 4, 5 ), biasanya kalo kita kirimkan parameter begitu kita bisa tangkap kedalam argument misalnya saya tulis ( a, b ), nah kalo saya tulis seperti itu Maka yg akan diambil hanya nilai pertama dan nilai ke-2 saja sisamya engga jadi kalo kita tulis return disitu, gini :

    function myFunc(*a*, *b*) {

        return *a*;

    }

    console.log(myFunc(1, 2, 3, 4, 5,));

Maka hasilnya kalo yg di return a itu angka 1 dn kalo return b pasti aja hasilnya angka 2. Nah sekarang kalo kita tuliskan disitu parameter berikut nya tapi bukan parameter biasa, tapi saya tuliskan kasih titik 3 lalu nama variable nya bebas ya misalkan saya namakan myArgs, maka nanti saya bisa ambil sisanya dari parameter yg dikirimkan, nama nya juga rest ya sisa nya dari parameter yg dikirimin akan masuk kedalam variable myArgs yg bentuk nya array, gini :

    function myFunc(*a*, *b*, ...*myArgs*) {

        return *myArgs*;

    }

    console.log(myFunc(1, 2, 3, 4, 5,));

Kalian bisa lihat bahwa myArgs isinya adalah array dari sisa parameter yg dikirmkan, jadi kalo misal saya gabung saya kasih nilai gini :

function myFunc(*a*, *b*, ...*myArgs*) {

        return `a = ${*a*}, b =${*b*}, myArgs = ${*myArgs*}`;

    }

    console.log(myFunc(1, 2, 3, 4, 5,));

Atau misalkan kalo langsung saya tangkap semuanya juga bisa, gini :

function myFunc(...*myArgs*) {

*// return `a = ${a}, b =${b}, myArgs = ${myArgs}`;*

        return *myArgs*;

    }

    console.log(myFunc(1, 2, 3, 4, 5,));

Jadi nanti hasilnya adalah seluruh parameter yg dikirimkan yg hasilnya berbentuk array, nah rest parameter ini hanya bisa kita gunakan di akhir dari argument nya jadi gabisa kalo dia yg pertama terus baru argument berikutnya, gini misal :

function myFunc(...*myArgs*, *a*, *b*) {

        return *myArgs*;

    }

    console.log(myFunc(1, 2, 3, 4, 5,));

Nah kaya gitu gabisa karena kan myArgs nya udah ngambil seluruh parameternya jadi argument a dan b nya mau nangkep apa gitu. Jadi rest parameter harus diakhir dari formal parameter.

Selanjut nya kalo misalkan kalian inget didalam function itu sebetulnya kita secara default punya variable namanya arguments, nah arguments ini otomatis berisi semua parameter yg dikirim pada saat fungsi dipanggil, gini :

function myFunc() {

*// return `a = ${a}, b =${b}, myArgs = ${myArgs}`;*

*// return myArgs;*

        return **arguments**;

    }

    console.log(myFunc(1, 2, 3, 4, 5,));

Nah arguments itu akan berisi semua yg dikirimkan jadi sebener nya bisa juga, tapi arguments itu bukan array, kalo bukan array kita gabisa pake method” yg array punya. Kalo misalkan kita ganti jadi rest parameter, gini :

    function myFunc(...*myArgs*) {

        return *myArgs*;

    }

    console.log(myFunc(1, 2, 3, 4, 5,));

Nah ini sama juga ngambil semua parameternya tapi bentuk nya sekarang udah array jadi kalo kalian pengen hasilnya array bisa pake rest parameter, kalo kalian ga pengen pake rest parameter bisa juga pake arguments tapi kita harus ubah dulu arguments nya jadi array, kita bisa ubah pake Array.from(), gini :

    function myFunc() {

*// return `a = ${a}, b =${b}, myArgs = ${myArgs}`;*

*// return myArgs;*

        return Array.from(**arguments**);

    }

    console.log(myFunc(1, 2, 3, 4, 5,));

Nah kalo gitu udah jadi array. Atau kita juga udah belajar spread Operator jadi kita bisa kasih return lalu arguments nya kita bisa masukan kedalam spread operator nya, gini :

    function myFunc() {

        return [...**arguments**];

    }   console.log(myFunc(1, 2, 3, 4, 5,));

Jadi kalo saya jalanin dia bentuknya betul-betul array, jadi banyak macem cara yang tujuan nya sama ya silahkan pilih yg enaknya gimana.

Misalkan kita coba kasus nya kita mau menjumlahkan seluruh parameter yg dikirimkan jadi saya pengen jumlahin 1+2+3+4+5, misalkan gini :

    function jumlahkan() {

    }

    console.log(jumlahkan(1, 2, 3, 4, 5,));

Nah ini kalo kita pake rest parameter kita bisa tangkep dulu semua isi nya misalnya kita kasih angka yg jadinya array, kita bisa gunakan for biasa misalkan kita bisa minta bantuan variable baru, kita bisa gunakan for biasa atau for..of.

Nah kalo kalian tau for..of itu cara penulisan nya pertama nya tulis variable yg nantinya variable itu merepsentasikan isi array nya, gini jadinya :

function jumlahkan(...*angka*) {

    let total = 0;

    for (const a of *angka*) {

        total += a;

    }

    return total;

}

console.log(jumlahkan(1, 2, 3, 4, 5,));

Kalo saya jalanin hasilnya 15 gaada masalah ya, jadi dengan menggunakan rest parameter bisa seperti itu, atau kita mau ringkas pengulangan nya juga bisa ya, kalian mau pake Higher order function misalnya reduce(), gini :

    function jumlahkan(...*angka*) {

        return *angka*.reduce((*a*, *b*) => *a* + *b*);

    }

    console.log(jumlahkan(1, 2, 3, 4, 5,));

Jadi simpel kita bisa manfaatin rest parameter untuk kasus kita yang ini. Nah kita juga bisa melakukan rest parameter pada saat kita melakukan array destructuring. Misalnya saya punya sebuah array namanya kelompok1 misalkan kelompok isi nya nama” orang, gini :

const kelompok1 = ['Sandhika', 'Doddy', 'Erik', 'Fajar', 'Hendra'];

Misalnya kalian mau atur supaya setiap nama yang muncul pertama dia adalah ketua, lalu setiap nama yang muncul ke-2 dia adalah wakil ketua sisanya adalah anggota, nah sebenernya bisa kalian tulis gini ya ( const ketua = kelompok1[0]; ), terus bikin lagi buat wakil nya di isi kelompok1 index ke-1.

Nah kalo kita pake distructuring ini bisa lebih simpel lagi, kita langsung aja bikin const lalu array distructuring, gini :

    const kelompok1 = ['Sandhika', 'Doddy', 'Erik', 'Fajar', 'Hendra'];

    const [ketua, wakil, ...anggota] = kelompok1;

jadi yg pertama ketua ke-2 wakil lalu sisanya dan seterusnya anggota, jadi kita bisa pake array distructuring juga bisa.

**// OBJECT DISTRUCTURING**

Object distructuring juga bisa misalkan saya punya variable namanya team, misalnya ini team developer nya yg terdiri dari projeck manejement saya singat pm, dan yg lain nya juga, gini :

    const team = {

        pm: 'Sandhika',

        frontEnd1: 'Doddy',

        frontEnd2: 'Erik',

        backEnd: 'Fajar',

        ux: 'Hendra',

        devOps: 'Ferry'

    }

Nah kalo kalian mau pisahkan menggunakan distructuring, misalkan kalian mau pisahkan antara pm dan sisa team nya, kalian bisa gini :

    const team = {

        pm: 'Sandhika',

        frontEnd1: 'Doddy',

        frontEnd2: 'Erik',

        backEnd: 'Fajar',

        ux: 'Hendra',

        devOps: 'Ferry'

    }

    const { pm, ...myTeam } = team;

    console.log(pm);

Jadi kalo kita console pm itu pasti ‘Sandhika’ ya, tapi kalo kita console myTeam yg muncul nya sisanya.

**// Filtering**

Dan kalo kalian mau ngelakuin filtering pada parameter yang ada ya, parameter yg dikirimkan, in contoh kasus aja ya, misalkan saya punya function nmanya filterBy() terus saya panggil functionnya terus misalkan saya mau ngefilter nilai-nilai berdasarkan tipe data nya nanti nilai-nilainya saya kirim kan pada argument itu.

Contoh nya kalo saya mau mencari angka pada nilai-nilai berikut ini :

    function filterBy() {

    }

    console.log(filterBy('number', 1, 2, 'Sandhika', false, 10, true, 'Doddy'));

Nah saya mau memilih mana aja dari parameter itu yang bentuknya angka, berarti saya akan bagi yg pertama tipenya mau apa lalu sisanya nilainya, jadi disitu saya punya 2 parameter yg pertama type nya apa yg ke-2 argument nya atau values nya lah.

Jadi sebenernya kalian bisa looping satu persatu nilai dari values nya tapi kalo mau langsung kalian bisa pake sebuah function namanya filter(), gini :

Jadi v disitu kita anggap isi dari values nya dan v itu akan mengembalikan nilai kita cek typeof nya typeof ini buat cek tipe data nya apa ya, jadi kalo tipe data dari v nya itu sama dengan number, kan yg awal itu kita kasih tau type nya ‘number’, ya berarti lagi nyari tipe data nya number. Jadi kalo tipe data dari isi array values itu ada tipe data number maka return lah itu.

Nah itukan nyari yg tipe data nya number, terus kalo mau nyari tipe data string ganti aja ‘number’ nya jadi ‘string’, gini :

        function filterBy(*type*, ...*values*) {

            return *values*.filter(*v* => typeof *v* === *type*);

        }

        console.log(filterBy('string', 1, 2, 'Sandhika', false, 10, true, 'Doddy'));

Nah maka dia akan ngecek typeof nya yg string maka yg tampil ‘Sandhika’ dan ‘Doddy’ aja karena dia aja yg string. Dan enak nya kalo pake filter dia langsung mengembalikan menjadi array.