**Destructuring an array**

const *array* = [1,2,3];  
const [*a*, *b*, *c*] = *array*;  
*console*.log(*a*,*b*,*c*);

Deci, putem declara mai multe variabile intre [] si fiecare varianila va corespunde unui index din array

* Daca punem mai multe variabile intre [] ca valori in array, variabilele ce nu au un element in array vor fi undifined
* Daca de ex am vrea elementele de pe index 0 si 2, fara 1, lasam o gaura intre variabile, adica [a, , b]
* const *array* = [1,2,3];  
  const [*a*, , *c*] = *array*;  
  *console*.log(*a*,*c*);
* Prin [] putem si schimba valorile la 2 variabile sau mai multe cu locul foarte usor:
* a = 10;  
  b = 20;  
    
  [a, b] = [b, a]
* Daca avem array in array, putem face destucturare in destructurare:
* const *array* = [1,2, [3,4]];  
    
  const [*a*,*b*,[*c*,*d*]] = *array*;  
    
  *console*.log(*a*,*b*,*c*,*d*);
* Se poata intampla sa avem mai multe variabile ca elemente in array, dar le putem seta valori default:
* const *array* = [1,2];  
    
  const [*a* = 0,*b* = 0,*c* = 0] = *array*;  
    
  *console*.log(*a*,*b*,*c*);

**Destructuring objects**

* O facem prin a folosi { } si intre ele punem numele la fielduri! Atentie, variabilele vor avea deci nume identice cu fieldurile, si asa nu conteaza ordinea
* const *obj* = {  
   firstName : 'Eduard',  
   age : 21,  
   country: 'MDA'  
  }  
    
  const {*firstName*, *age*, *country*} = *obj*;  
    
  *console*.log(*firstName*, *age*, *country*);
* Daca vrem alte nume, dupa fielduri punem : si numele ales la variabile
* const *obj* = {  
   firstName : 'Eduard',  
   age : 21,  
   country: 'MDA'  
  }  
    
  const {firstName :*fn*, age :*ag*, country :*cnt*} = *obj*;  
    
  *console*.log(*fn*, *ag*, *cnt*);
* Putem pune si valori default, in caz ca asa field nu e gasit
* const *obj* = {  
   firstName : 'Eduard',  
   age : 21,  
   country: 'MDA'  
  }  
    
  const {*firstName* = 'test', age :*ag* = 18, *country* = 'nothing'} = *obj*;  
    
  *console*.log(*firstName*, *ag*, *country*);
* Daca vrem sa atribuim valori unor variabile deja existente, folosim ()
* const *obj* = {  
   firstName : 'Eduard',  
   age : 21,  
   country: 'MDA'  
  }  
    
  let *firstName*;  
  let *age*;  
  let *country*;  
    
  ({*firstName*, *age*, *country*} = *obj*);
* Daca avem nested obj in obj folosim : { } la acel field ce e object

const *obj* = {  
 firstName : 'Eduard',  
 address: {  
 code: 2091,  
 country: 'MD'  
 }  
}  
  
const {*firstName*, address: {*code*, *country*}} = *obj*;

* O functie poate sa aiba si un parametru format din decomposition

const *obj* = {  
 firstName: 'Eduard',  
 age : 20,  
 code: 5210  
}  
  
function test({code, age, firstName}){  
 *console*.log(firstName, age, code);  
}  
  
test(*obj*);

Putem sa le punem si valori default la fel.

**Spread operator(...)**

* Daca vrem sa mutam toate elementele unui array in alt nou array ce mai contine elemente, o putem face foarte usor cu ...
* const *array* = [10, 20, 30, 40];  
    
  const *newArray* = [1,2,3,...*array*];  
    
  *console*.log(*newArray*);

asa, JS va sti ca newArray nu va avea pur si simplu o referinta la array, ca simplu element din array, ci va copia toate elementele sale. Va face copii!!!

Acest operator ia valorile din acest array si le scrie el de mana ca si cum

* Daca vom folosi un asemenea argument, cu ..., ar fi ca si cum JS ar lua si ar scrie de mana fiecare element:
* const *array* = [10, 20, 30, 40];  
    
  *console*.log(...*array*);

e echivalent cu console.log(10,20,30,40);

* ... e bun si pentru a face o copie la array cu [...oldArray] si pentru a da merge la 2 array

const *array1* = [10, 20, 30, 40];  
const *array2* = [1, 2, 3, 4];  
const *merged* = [...*array1*,...*array2*]  
*console*.log(*merged*);

* **... functioneaza cu orice este iterable, nu doar cu array!Si ele sunt:**

**- string**

**- array**

**- map**

**- set**

**Numai nu obiecte simple!**

const *name* = 'Eduard';  
*console*.log([...*name*])

**[ 'E', 'd', 'u', 'a', 'r', 'd' ]**

* Evident, asa ceva e invalid:

` ${...array}



* Din versiunea 2023, ... merge si pe obiecte, si anume e folosit pentru a include toate field:value intr-un nou obiect
* const *obj* = {  
   name: 'Eduard',  
   age: 20  
  }  
    
  const *newObj* = {...*obj*,country: 'MDA'}  
    
  *console*.log(*newObj*);

Iarasi, face doar copii!!

**Rest pattern(...) and parameters**

* Rest pattern tot foloseste ..., doar ca el face inversul la ce face spread operator, adica el impreuneaza elementele ramase intr-un array
* De asta si se numeste rest, caci el ia restul elementelor ramase si le pune intr-un array nou

const *array* = [1,2,3,4,5]  
  
const [*a*,*b*,...*rest*] = *array*;  
  
*console*.log(*a*,*b*,*rest*);

1 2 [ 3, 4, 5 ]

**Atentie! ... trebuie sa fie mereu ultimul in [] din stanga!!!**

const [*a*,*b*,...rest, *cant*] = [10,20,30,40,50,...*array*];



* Si asa ceva e posibil:
* const *array* = [1,2,3,4,5]  
    
  const [*a*,*b*,...*rest*] = [10,20,30,40,50,...*array*];  
    
  *console*.log(*a*,*b*,*rest*);
* Poate fi folosit si cand preluam proprietatile de la un obiect, pentru a crea un nou obiect cu proprietatile ramase:

const *obj* = {  
 name: 'Eduard',  
 age: 21,  
 country: 'MDA',  
 code: 2092  
}  
  
const {*name*, ...*address*} = *obj*;  
  
*console*.log(*address*);

{ age: 21, country: 'MDA', code: 2092 }

* Putem saa il folosim si la ultimul parametru al metodei, care va fi un array ce va lua toti ceilalti parametri
* function test(main, ...rest){  
   *console*.log(main, rest);  
  }  
    
  const *array* = [1,2,3,4,5];  
  test(...*array*);

1 [ 2, 3, 4, 5 ]

**Short circuiting**

* Desi || si && se folosesc cu boolean, pot fi folosite si nu direct pe boolean
* Pot fi folosite pe orice obj sau primitiva
* Ele returneaza acel obiect care este evaluat ca true, sau daca nici unul nu e true, ultimul e returnat
* De ex, || returneaza primul element care este evaluat ca true. Aici ambele ar fi evaluate ca true, deci se returneaza 3

3 || ‘test’

* Dar, evident, ‘’ || 10 va returna 10
* Dar, daca nici unul nu e true, ultimul mereu e returnat, ca e true sau false

*console*.log('' || null || undefined);

undefined

* && lucreaza invers ca ||. Daca || returneaza prima valoare true, && returneaza prima valoare false:

*console*.log(0 && null && 10);

0

* Daca toate sunt true, o returneaza pe ultima

**Nullish operator(??)**

* problema cu || si && e ca ‘’ si 0 tot sunt luate ca false, si asta cam poaet cauza uneori probleme
* ?? - Returneaza primul element care nu este **null** sau **undefined**
* *console*.log(0 ?? 10);

10

*console*.log(null ?? 10);

10

* este foarte bun pentru a ne asigura ca parametrii nu sunt null sau undefined
* function set(value){  
   value = value ?? 0;  
    
   *console*.log(value);  
  }  
    
  set(null);  
  set();

0

0

**Logical assignment operators(||= si &&= si ??=)**

* ||= si &&= si ??= sunt utilizate pentru a evita repetarile de genul:
* const *rest1* = {  
   name: 'test1',  
   guests: 20  
  }  
    
  const *rest2* = {  
   name: 'test1',  
   country: 'MD'  
  }  
    
  *rest1*.guests = *rest1*.guests || 10;  
  *rest2*.guests = *rest2*.guests || 10;  
  *console*.log(*rest1*, *rest2*);

Aici verificam daca rest1 si rest2 au o proprietate guests sau daca o au, sa nu fie null/undefined sau 0.

* Totusi, ar fi mai comod sa scriem asa:

*rest1*.guests ||= 10;  
*rest2*.guests ||= 10;

Sau, daca vrem doar sa nu fie null sau undefined

*rest1*.guests ??= 10;  
*rest2*.guests ??= 10;

**array.entries()**

* Orice array, doar array, are un iterator creat, si metoda entries() returneaza acest iterator
* Iteratorul contine cate un obiect pentru fiecare element, unde stocheaza indexul sau si valoarea sa:

const *array* = [10,20,30,40,50];  
  
for(const i of *array*.entries())  
 *console*.log(i);

[ 0, 10 ]

[ 1, 20 ]

[ 2, 30 ]

[ 3, 40 ]

[ 4, 50 ]

const *array* = [10,20,30,40,50];  
  
for(const [index, value] of *array*.entries())  
 *console*.log(index+" "+value);

**Enhanced object literals**

* Cand avem un obiect ce are in el mai multe obiecte ca field, am putea sa cream obiectele interne in afara,gen in loc de:
* const *obj* = {  
   name: 'Eduard',  
   address: {  
   country: 'MD',  
   code: 2092  
   }  
  }

putem face asa:

const *address* = {  
 country: 'MD',  
 code: 2092  
}  
  
const *obj* = {  
 name: 'Eduard',  
 address : *address*}

Totusi, sa repetam address: address e cam enervant

* In noile versiuni de JS, nu mai e necesar sa repetam asta, o scriem doar odata si JS va avea grija sa o mai repete odata:
* const *address* = {  
   country: 'MD',  
   code: 2092  
  }  
    
  const *obj* = {  
   name: 'Eduard',  
   *address*,  
  }
* Tot in noile versiuni de JS, putem defini o metoda mai usor, fara function si fara :
* const *obj* = {  
   name: 'Eduard',  
   func: function() { }  
  }

const *obj* = {  
 name: 'Eduard',  
 func() { },  
 func2() { }  
}

* Tot in noile versiuni de JS, putem defini numele unei proprietati dinamic, folosind []

const *fieldName* = 'name'  
  
const *obj* = {  
 [*fieldName*]: 'Eduard',  
}

{ name: 'Eduard' }

Putem folosi si [`${}`]

**Optional Chaining (?.)**

* ?. face ca daca o proprietate nu exista, sa fie returnat undefined, si asa se evita posibilele exceptii
* Daca o proprietate e null sau undefined, e logic ca ea nu poate sa fie un obiect
* Daca apelam o proprietate din undefined, primim exceptie:
* const *obj* = {  
   name: 'Eduard',  
   address: {  
   country: 'MD',  
   code: 2092  
   }  
  }  
    
  *console*.log(*obj*.address.country.city.population);

city nu exista un address, si e logic ca avem undefined.population, si primim asta:

TypeError: Cannot read properties of undefined (reading 'population')

* O solutie ar fi sa verificam cu if daca proprietatile date exista:

if(*obj*.address && *obj*.address.country && *obj*.address.country.city)  
 *console*.log(*obj*.address.country.city.population);

* Dar, asta poate fi simplificat cu ?.

*console*.log(*obj*.address?.country?.city?.population);

undefined

Deci, odata ce ?. a vazut ca city e undefined, a returnat udnefined, fara sa mai incerce sa cauta ceva dupa city

* Merge si pentru metode
* *console*.log(*obj*.address?.country?.getName?.() ?? 'Method doesn\'t exists');
* Merge si la array sa vada daca un element la asa index exista
* const *array* = [{name: 'eduard'}]  
    
  *console*.log(*array*[3]?.name ?? 'Element with index 3 doesn\'t exist');

**Object methods**

**Atentie, doar array au default metodele .keys(), .values(), .entries()! Obiectele nu le au si primim eroare de le apelam!!! Arrays sunt automat iterable,dar nu si obiectele noastre! Doar obiectele ce sunt iterable au asta**

const *obj* = {}  
  
*console*.log(*obj*.values())



* **.keys(obj)** – returneaza un iterbale cu fields ale unui obiect sau index ale unui array

const *obj* = {  
 name: 'Eduard',  
 age: 21,  
 country: 'MDA'  
}  
  
*console*.log(*Object*.keys(*obj*));

[ 'name', 'age', 'country' ]

const *array* = [  
 {  
 name: 'Eduard',  
 age: 21  
 },  
 {  
 name: 'Alex',  
 age: 21  
 },  
 {  
 name: 'Sanea',  
 age: 20  
 }  
]  
  
*console*.log(*Object*.keys(*array*));

[ '0', '1', '2' ]

* **.values(obj)** – returneaza un iterbale toate valorile fieldurilor ale unui obiect sau valorile unui array
* const *array* = [  
   {  
   name: 'Eduard',  
   age: 21  
   },  
   {  
   name: 'Alex',  
   age: 21  
   },  
   {  
   name: 'Sanea',  
   age: 20  
   }  
  ]  
    
  *console*.log(*Object*.values(*array*));

[

{ name: 'Eduard', age: 21 },

{ name: 'Alex', age: 21 },

{ name: 'Sanea', age: 20 }

]

const *obj* = {  
 name: 'Sanea',  
 age: 20  
 }  
  
  
*console*.log(*Object*.values(*obj*));

[ 'Sanea', 20 ]

* **.entries(obj)** – returneaza un iterbale pentru fiecare field al obiectului, un array ce contine [field, value], deci returneaza un array cu arrays in el
* const *obj* = {  
   name: 'Eduard',  
   age: 21,  
   country: 'MDA',  
  }  
    
  *console*.log(*Object*.entries(*obj*));

[ [ 'name', 'Eduard' ], [ 'age', 21 ], [ 'country', 'MDA' ] ]

**for of and for in**

* for of poate fi folosit pe array si obiectele iterable:
* const *obj* = {name: 'eduatd'}  
    
  for(const a of *obj*)  
   *console*.log(a);

ERROR

* for in poate fi folosit si pe obiectele ce nu sunt iterable
* const *obj* = {name: 'eduatd'}  
    
  for(const a in *obj*)  
   *console*.log(a);

name