```
//DESTRUCTURING:
// O operador "destructuring" é um operador do Javascript criado á partir da versão ECMA Script 06, de 2015. A proposta do destructuring é justamente desestruturar elementos iteráveis de uma forma mais dinâmica que as formas convencionais. Por exemplo, em uma única linha de código é possível capturar todos os valores de um objeto sem usar notação ponto, criar variáveis fora do contexto do objeto e atribuir os mesmos valores capturados do objeto ás variáveis que acabamos de criar, e tudo isso em 1 única linha! Impressionante não?
//O destruturing é um operador que pode ser usado com facilidade tanto em objects quanto em arrays, porém, para cada um deles temos que usar sintaxes diferentes. Para objects usamos o destructuring por através das chaves, enquanto para arrays usamos o destructuring por através dos colchetes.
//Vejamos alguns exemplos e aplicações:

//DESTRUTURIZANDO UM OBJETO:
const pessoa = { //Perceba aqui que criamos um objeto para uma pessoa...
    nome: 'Ana',
    idade: 5,
    endereco: {
        logradouro: 'Rua ABC',
        numero: 1000
    }
}

const {nome, idade} = pessoa;//Note como destruturizamos valores do objeto e ainda por cima atribuímos eles a variáveis fora do contexto do objeto...
/*
    Explicando o código acima:

        - const: Usamos const para definir que as variáveis criadas na desestruturização seriam constantes, mas poderíamos
                    usar var ou let se quiséssemos;

        - { }: Os colchetes são o destructuring em si, quando usado sem seguida após um const, var ou let, o Javascript já
                sabe que é um destructuring para um object, se fosse um array teríamos usado colchetes;

        - nome, idade: sempre devemos usar os mesmos nomes de chaves do objeto que desejamos desestruturar para capturar seus
                        valores, esses nomes acabarão se tornando variáveis dentro do contexto onde o destructuring foi utilizado. Mas não precisamos utilizar os mesmos nomes das chaves para definir nossas variáveis destructuring, podemos modificar esses nomes usando a seguinte sintaxe: "nome: nome_Novo";
```

```
        - =: O operador de atribuição atribuí as chaves retiradas a um identificador de objeto, como o nosso identificador era
                o objeto "pessoa", usamos o mesmo nome para saber que vamos retirar os valores desse objeto em questão;

        - pessoa: o nome do identificador que desejamos desestruturar;
*/
console.log('1)', nome, idade); //Veja que embora as constantes criadas no destructuring não existam literalmente no código, elas pod
em ser chamadas como se existissem. Se tentássemos imprimí-las no console antes do destructuring teríamos um erro...



//USANDO O NOME QUE QUISERMOS AS VARIÁVEIS DESESTRUTURADAS:
const {nome: n, idade: age} = pessoa; //Perceba que retiramos os valores das chaves "nome" e "idade", mas damos as variáveis criadas
com destructuring nomes em inglês;
console.log('2)', n, age); //Perceba que vamos conseguir retirar os valores do objeto "pessoa" normalmente...



//O QUE ACONTECE SE TENTAMOS REFERENCIAR UMA CHAVE QUE NÃO EXISTE DENTRO DE UM OBJETO?
const {sobrenome} = pessoa; //a chave "sobrenome" não existe em pessoa...
console.log('3)', sobrenome); //Perceba que o programa não dá erro, o Javascript apenas apresenta undefined;



//ATRIBUINDO UM VALOR PADRÃO A UMA VARIÁVEL DESTRUCTURING CASO ELA NÃO TENHA VALOR NENHUM OU NÃO EXISTA DENTRO DE UM OBJETO:
const {bemHumorada = true} = pessoa; //Perceba que a chave "bemHumorada" também não existe no objeto pessoa, mas atribuímos um valor
padrão para impedir que o Javascript apresente "undefined"...
console.log('4)', bemHumorada); //Veja que em vez de undefined ele apresenta o valor padrão "true"...



//DESESTRUTURANDO CHAVES QUE ESTÃO DENTRO DE ESTRUTURAS ANINHADAS:
const {endereco: {logradouro, numero}} = pessoa; //No objeto "pessoa" temos uma estrutura aninhada dentor de "endereco" para acessá-
lo veja que abrimos caminho usando ":" e abrindo a estrutura aninhada com as "{ }", dentro da estrutura usamos os nomes das chaves ex
istentes para capturar os valores das chaves aninhadas em "endereco"...
console.log('5)', logradouro, numero); //Perceba que conseguimos pegar os valores corretamente...
```

```
//BUSCAS EM ESTRUTURAS ANINHADAS ONDE O ELEMENTO PAI NÃO EXISTE GERAM ERRO:
const {endereco: {complemento: {detalhes}}} = pessoa; //Perceba que dentro de endereco não existe uma chave "complemento" e nem uma c
have "detalhes" dentro de uma chave complemento, ou seja, temos uma situação onde uma estrutura "pai" que é "complemento" não existe,
 nestes casos o Javascript não irá apresentar "undefined", ele vai dar erro mesmo. "Undefined" só é apresentado quando a estrutura in
existente dentro de um objeto é um elemento "filho"...
console.log(detalhes); //Perceba a falha ao tentar encontrar detalhes dentro de um elemento pai inexistente...



//DESESTRUTURIZANDO UM ARRAY:
const alfabeto = ['a', 'b', 'c']; //Perceba que o array, diferente do object não possuí nomes de chaves, por isso para criar variávei
s desestruturadas sobres os índices do array temos que criar variáveis na sequência dos valores que desejamos retirar...
const [primeiroValor, segundoValor, terceiroValor] = alfabeto; //Veja que simplesmente fomos criando nomes que referenciam diretament
e as posições dos índices...
console.log('6)', terceiroValor, segundoValor, primeiroValor);



//CRIANDO UM ARRAY DINÂMICO USANDO DESTRUCTURING:
const [a, b, c] = [0, 1, 2] //Perceba que não existe nenhum array sendo desestruturado, estamos usando o operador destructuring para
gerar um array com números de 0 a 2, que irá atribuir seus números as variáveis a, b e c...
console.log('7)', a, b, c); //Note que embora o array não exista literalmente, as variáveis armazenam os valores passados em ordem pe
lo array atribuído ao destructuring...



//PODEMOS PULAR ÍNDICES DO ARRAY:
const [n1, , n3, n4, , n6] = [0, 1, 2, 3, 4, 5]; //Perceba que criamos variáveis pulando alguns valores do array usando a vírgula...
console.log('8)', n1, n3, n4, n6); //Elas pegarão exatamente a sequência dos valores pulados...



//ATRIBUINDO UM VALOR PADRÃO A UM ARRAY INEXISTENTE:
```

```
const [val1, val2 = 'Não há valor'] = []; //Passamos um array vazio, veja que estamos tentando captura 2 valores, o primeiro por não ter valor padrão vai gerar "undefined", o segundo irá apresentar o seu valor padrão...
console.log('9)', val1, val2);



//PEGANDO VALORES EM ARRAYS ANINHADOS:
const [ , [ , nota]] = [[ , 8, 8], [9, 6, 8]]; //Perceba que temos um array com 2 arrays aninhados dentro dele, o no destructuring queremos pegar o 2 elemento do segundo array usando a variável "nota"...
console.log('10)', nota); //Perceba que conseguimos pegar o elemento naturalmente...



//USANDO DESTRUCTURING NUM OBJETO USANDO FUNÇÃO:
function rand({min = 0, max = 1000}) { //Perceba que o destructuring é usando dentro do campo de parâmetros da função, sua missão é desestruturizar 2 valores min e max que forem passados como objeto para os parâmetros da função...
    const valor = Math.random() * (max - min) + min; //Usamos a função random para gerar um número aleatório entre o valor mínimo e o valor máximo passado. Para mais detalhes sobre a função random veja a aula 45, mas o importante é saber o random vai gerar um valor entre quis quer números entre o valor mínimo e o máximo que forem passados, se nenhum valor for passado ele vai gerar um número aleatório entre 0 e 1000...
    return Math.floor(valor); //Como random trabalha com casas decimais usamos uma função Math.floor para arredondar o valor para baixo transformando-o num inteiro...
}
const obj = {max: 50, min: 40}; //aqui criamos um objeto com valores para max e min...
console.log('11)', rand(obj)); //e passamos esse objeto para a função rand, onde os valores de max e min serão desestrururizados...
console.log('12)', rand({min: 955})); //Podemos passar um único valor de objeto, já que a função já em valor padrão, criando um objeto sem identificador dentro da própria passagem de parâmetros...
console.log('13)', rand({})); //Podemos não passar objeto valor nenhum, pois a função já possuí valores padrão...
console.log(rand()); //O que não podemos fazer é não passar objeto nenhum, pois a função possuí um destructuring dentro dos seus parâmetros, se não passarmos pelo menos as chaves que identificam um elemento como colchetes, ela não conseguirá desestruturizar valor nenhum, gerando erro na função...



//USANDO DESTRUCTURING NUM ARRAY USANDO FUNÇÃO:
function randArray([min = 0, max = 1000]) { //Perceba que nessa função vamos usar o destructuring 2 vezes, a primeira é nessa linha mesmo, onde iremos desestruturas os valores passados por um array qualquer, em valores mínimo e máximo...
```

```
    if (min > max) [min, max] = [max, min]; //Aqui estamos desestruturando mais uma vez ao fazer um desvio condicional "if", onde, caso o usuário digite um valor mínimo que seja maior que o valor máximo, usamos o destructuring para gerar novas variáveis "min" e "max" que trocam o que for min pelo max e o que for max pelo min...
    const valor = Math.floor(Math.random() * (max - min) + min); //Depois só pegamos os números randômicamente como fizemos no exemplo com os objetos...
    return (valor);
}
console.log('14)', randArray([50, 40])); //Note que não especificamos quem é min e quem é max, pois arrays não possuem chave e valor, somos obrigados a passar os valores segundo a posição índice para serem capturados no lugar certo, mas como o nosso laço "if" na função troca um valor mínimo que seja maior pelo máximo, podemos passar valores em qualquer ordem despreocupamente...
console.log('15)', randArray([993])); //Podemos passar apenas um valor, pois a função já possuí valores padrão e vai fazer o teste para ver se o valor passado deve ser máximo ou mínimo...
console.log('16)', randArray([])); //Podemos passar um array vazio, pois a função já possuí valores padrão para mínimo e máximo...
console.log(randArray()); //Não podemos somente chamar a função, pois o campo de parâmetros dela tem um destructuring, e não podemos desestrururizar nada...
```

RESULTADO NO CONSOLE...

```
[Running] node "c:\Users\User\Documents\JAVASCRIPT\ARQUIVOS_DAS_AULAS\044-Manipulando_Destructuring.js"
1) Ana 5
2) Ana 5
3) undefined
4) true
5) Rua ABC 1000
6) c b a
7) 0 1 2
8) 0 2 3 5
9) undefined Não há valor
10) 6
11) 40
12) 967
13) 642
14) 44
15) 994
16) 845
```

RESULTADO PARA FALHA AO CHAMAR CHAVE PAI QUE NÃO EXISTE DENTRO DE UM OBJECT...

```
[Running] node "c:\Users\User\Documents\JAVASCRIPT\ARQUIVOS_DAS_AULAS\044-Manipulando_Destructuring.js"
1) Ana 5
2) Ana 5
3) undefined
4) true
5) Rua ABC 1000
c:\Users\User\Documents\JAVASCRIPT\ARQUIVOS_DAS_AULAS\044-Manipulando_Destructuring.js:63
const {endereco: {complemento: {detalhes}}} = pessoa; //Perceba que dentro de endereco não existe uma chave "complemento" e nem uma c
have "detalhes" dentro de uma chave complemento, ou seja, temos uma situação onde uma estrutura "pai" que é "complemento" não existe,
 nestes casos o Javascript não irá apresentar "undefined", ele vai dar erro mesmo. "Undefined" só é apresentado quando a estrutura in
existente dentro de um objeto é um elemento "filho"...
                              ^

TypeError: Cannot read property 'detalhes' of undefined
    at Object.<anonymous> (c:\Users\User\Documents\JAVASCRIPT\ARQUIVOS_DAS_AULAS\044-Manipulando_Destructuring.js:63:33)
    at Module._compile (internal/modules/cjs/loader.js:1072:14)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1101:10)
    at Module.load (internal/modules/cjs/loader.js:937:32)
    at Function.Module._load (internal/modules/cjs/loader.js:778:12)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:76:12)
    at internal/main/run_main_module.js:17:47

[Done] exited with code=1 in 0.143 seconds
```

RESULTADO DE FALHA AO TENTAR CHAMAR UMA FUNÇÃO QUE POR PARÂMETRO DESESTRUTURA OBJECTS, SEM PASSAR NENHUM OBJETO NA CHAMADA DA FUNÇÃO...

```
[Running] node "c:\Users\User\Documents\JAVASCRIPT\ARQUIVOS_DAS_AULAS\044-Manipulando_Destructuring.js"
1) Ana 5
2) Ana 5
3) undefined
4) true
5) Rua ABC 1000
6) c b a
7) 0 1 2
```

```
8) 0 2 3 5
9) undefined Não há valor
10) 6
11) 44
12) 994
13) 178
c:\Users\User\Documents\JAVASCRIPT\ARQUIVOS_DAS_AULAS\044-Manipulando_Destructuring.js:100
function rand({min = 0, max = 1000}) { //Perceba que o destructuring é usando dentro do campo de parâmetros da função, sua missão é d
esestruturizar 2 valores min e max que forem passados como objeto para os parâmetros da função...
              ^

TypeError: Cannot read property 'min' of undefined
    at rand (c:\Users\User\Documents\JAVASCRIPT\ARQUIVOS_DAS_AULAS\044-Manipulando_Destructuring.js:100:16)
    at Object.<anonymous> (c:\Users\User\Documents\JAVASCRIPT\ARQUIVOS_DAS_AULAS\044-Manipulando_Destructuring.js:108:13)
    at Module._compile (internal/modules/cjs/loader.js:1072:14)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1101:10)
    at Module.load (internal/modules/cjs/loader.js:937:32)
    at Function.Module._load (internal/modules/cjs/loader.js:778:12)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:76:12)
    at internal/main/run_main_module.js:17:47

[Done] exited with code=1 in 0.149 seconds
```

RESULTADO DE FALHA AO TENTAR CHAMAR UMA FUNÇÃO QUE POR PARÂMETRO DESESTRUTURA ARRAYS, SEM PASSAR NENHUM ARRAY NA CHAMADA DA FUNÇÃO...

```
[Running] node "c:\Users\User\Documents\JAVASCRIPT\ARQUIVOS_DAS_AULAS\044-Manipulando_Destructuring.js"
1) Ana 5
2) Ana 5
3) undefined
4) true
5) Rua ABC 1000
6) c b a
7) 0 1 2
8) 0 2 3 5
9) undefined Não há valor
10) 6
```

```
11) 46
12) 999
13) 360
14) 41
15) 996
16) 956
c:\Users\User\Documents\JAVASCRIPT\ARQUIVOS_DAS_AULAS\044-Manipulando_Destructuring.js:113
function randArray([min = 0, max = 1000]) { //Perceba que nessa função vamos usar o destructuring 2 vezes, a primeira é nessa linha m
esmo, onde iremos desestruturas os valores passados por um array qualquer, em valores mínimo e máximo...
                 ^

TypeError: undefined is not iterable (cannot read property Symbol(Symbol.iterator))
    at randArray (c:\Users\User\Documents\JAVASCRIPT\ARQUIVOS_DAS_AULAS\044-Manipulando_Destructuring.js:113:19)
    at Object.<anonymous> (c:\Users\User\Documents\JAVASCRIPT\ARQUIVOS_DAS_AULAS\044-Manipulando_Destructuring.js:121:13)
    at Module._compile (internal/modules/cjs/loader.js:1072:14)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1101:10)
    at Module.load (internal/modules/cjs/loader.js:937:32)
    at Function.Module._load (internal/modules/cjs/loader.js:778:12)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:76:12)
    at internal/main/run_main_module.js:17:47

[Done] exited with code=1 in 0.167 seconds
```