deixe pontos = [{x: 1, y: 2}, {x: 3, y: 4}];// uma variedade de
dois objetos de pontos
Seja [{x: x1, y: y1}, {x: x2, y: y2}] = pontos;//
Destruturado em 4 variáveis.
(x1 === 1 && y1 === 2 && x2 === 3 && y2 === 4) // => true
Ou, em vez de destruir uma variedade de objetos, poderíamos destruir um
Objeto de matrizes:
Let Points = {P1: [1,2], P2: [3,4]};// um objeto
com 2 adereços de matriz
Seja {p1: [x1, y1], p2: [x2, y2]} = pontos;//
Destruturado em 4 VARs
(x1 === 1 && y1 === 2 && x2 === 3 && y2 === 4) // => true
Sintaxe de destruição complexa como essa pode ser difícil de escrever e difícil para
Leia, e você pode estar melhor apenas escrevendo suas tarefas
explicitamente com código tradicional como let x1 = pontos.p1 [0];.
Compreendendo a destruição complexa
Se você se encontrar trabalhando com código que usa atribuições complexas de destruição, há um útil
regularidade que pode ajudá -lo a entender os casos complexos.Pense primeiro em um regular (único-
valor) atribuição.Depois que a tarefa for concluída, você pode pegar o nome da variável da esquerda
lado da tarefa e use -a como uma expressão em seu código, onde ele avaliará para qualquer
valor que você atribuiu.O mesmo se aplica à atribuição de destruição.O lado esquerdo do lado de um
A atribuição de destruição parece uma matriz literal ou um objeto literal (§6.2.1 e §6.10).Depois do
A tarefa foi realizada, o lado esquerdo da mão funcionará como uma matriz válida literal ou objeto literal
em outros lugares do seu código.Você pode verificar se escreveu uma tarefa de destruição corretamente por
Tentando usar o lado esquerdo do lado do campo de outra expressão de atribuição:
// Comece com uma estrutura de dados e uma destruição complexa
deixe pontos = [{x: 1, y: 2}, {x: 3, y: 4}];
Seja [{x: x1, y: y1}, {x: x2, y: y2}] = pontos;
// Verifique sua sintaxe de destruição de destruição
Seja pontos2 = [{x: x1, y: y1}, {x: x2, y: y2}];// pontos2 == pontos