

- Very sweet syntactic sugar.
- Object and array literals are used on the RHS of = in declarations or assignments for construction of JavaScript objects and arrays. Destructuring allows the use of similar notation on the LHS of = for accessing the elements of an object or array.
- Many modern programming languages have similar syntax.
- [MDN reference](#).

Basic Examples

```
> let [a, b] = [42, 22, 33, 44]
```

```
undefined
```

```
> [a, b]
```

```
[ 42, 22 ]
```

```
> {a, b} = { b: 42, c: 22, a: 33, d: 44 }
```

```
{ b: 42, c: 22, a: 33, d: 44 }
```

```
> [a, b]
```

```
[ 33, 42 ]
```

```
> function f([a, b], {c, d}) {  
  console.log(a, b, c, d);  
}
```

```
undefined
```

```
> f([33, 2, 44, 55], { a: 2, b: 3, c: 42 })
```

```
33 2 42 undefined
```

```
undefined
```

```
>
```

Array Destructuring Examples

```
> let [a, b] = [22, 42]
undefined
> [a, b]
[ 22, 42 ]
> [a, b] = [b, a] //exchange without temporary
[ 42, 22 ]
> [a, b]
[ 42, 22 ]
> [a, , , b] = [1, 2, 3, 4] //ignored values
[ 1, 2, 3, 4 ]
> [a, b]
[ 1, 4 ]
```

Array Destructuring Examples Continued

```
> let [ x, y = 99 ] = [42] //default value of 99 for y  
undefined  
> [x, y]  
[ 42, 99 ]  
> let [ x1, y1 = 99 ] = [42, 22] //default not used  
undefined  
> [x1, y1]  
[ 42, 22 ]  
>
```

Array Destructuring Examples Continued

```
[a, ...b] = [1, 2, 3, 4] //rest parameters
```

```
[ 1, 2, 3, 4 ]
```

```
> [a, b]
```

```
[ 1, [ 2, 3, 4 ] ]
```

```
> [a, b] = [a, ...b] //spreading b
```

```
[ 1, 2, 3, 4 ]
```

```
> [a, b]
```

```
[ 1, 2 ]
```

Object Destructuring Examples

```
let { p, q } = { p: 22, q: 42 }
```

```
undefined
```

```
> [p, q]
```

```
[ 22, 42 ]
```

```
{ p, ...rest } = { a: 22, p: 42, b: 33 } //rest params
```

```
{ a: 22, p: 42, b: 33 }
```

```
> [p, rest]
```

```
[ 42, { a: 22, b: 33 } ]
```

```
> {p = 33, q = 42 } = { q: 99, a: 44 } //default value
```

```
{ q: 99, a: 44 }
```

```
> [p, q]
```

```
[ 33, 99 ]
```

Object Destructuring Examples Continued

//var names different from property names

```
> { a: p, b: q } = { p: 1, a: 2, q: 3, b: 4 }  
{ p: 1, a: 2, q: 3, b: 4 }  
> [p, q]  
[ 2, 4 ]
```

//var names different from property names with defaults

```
{ a: p = 22, b: q = 99 } = { a: 11 }  
{ a: 11 }  
> [p, q]  
[ 11, 99 ]  
>
```

Combining Object and Array Destructuring

```
> { a: [p, ...q], b: c } = {a: [1, 2, 3], b: 42}
{ a: [ 1, 2, 3 ], b: 42 }
> [p, q, c]
[ 1, [ 2, 3 ], 42 ]
> [ { a, ...b}, c] = [ {a: 2, b: 9, x: 22}, { a: 1}]
[ { a: 2, b: 9, x: 22 }, { a: 1 } ]
> [a, b, c]
[ 2, { b: 9, x: 22 }, { a: 1 } ]
>
```


Function Parameters Destructuring

```
> function f({a, b}) { console.log(a, b); }
```

```
undefined
```

```
> f({x: 22, b: 2, a: 99, y:2})
```

```
99 2
```

```
undefined
```

```
> function f([a, ...b], { c: x, ...y }) {  
  console.log(a, b, x, y);  
}
```

```
undefined
```

```
> f([1, 2, 3, 4], { c: 42, d: 22, e: 44 })
```

```
1 [ 2, 3, 4 ] 42 { d: 22, e: 44 }
```

```
undefined
```

```
>
```

Function Options Parameter Without Destructuring

Common to have some kind of options parameter to a function, where each option has a default value which can be overridden by the caller.

//without destructured formal param

```
function doOutput(text, opts={}) {  
  const inFormat = opts.inFormat || 'text';  
  const outFormat = opts.outFormat || 'html';  
  const lineLength = opts.lineLength || 60;  
  ...  
}
```

```
...  
doOutput('123 the'); //all default options  
doOutput('123 the', { outFormat: 'latex' });
```

Function Options Parameter With Destructuring

//with destructured formal param

```
function nextToken(text, {  
  inFormat = 'text',  
  outFormat = 'html',  
  lineLength = 60} = {}) {  
  ...  
}  
...
```

//calls as before

Removing Defaults From Function

Using destructuring in function header ok for small number of options; otherwise it clutters up the function header. I often prefer the following idiom:

```
function nextToken(text, options={})  
  //Object.assign() copies properties from subsequent args  
  //into first arg with later args winning.  
  const opts = Object.assign({}, OPTIONS, options);  
  //destructure opts as necessary  
  ...  
}  
  
const OPTIONS = {  
  inFormat: 'text',  
  outFormat: 'html',  
  lineLength: 60,  
  ...  
};
```