The **for...of loop** is the most recent addition to the family of for loops in JavaScript.

It combines the strengths of its siblings, the **for loop** and the **for...in loop**, to loop over any type of data that is **iterable** (meaning it follows the [iterable protocol](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Iteration_protocols" \t "_blank) which we'll look at in lesson 3). By default, this includes the data types String, Array, Map, and Set—notably absent from this list is the Object data type (i.e. {}). Objects are not iterable, by default.

Before we look at the for...of loop, let’s first take a quick look at the other for loops to see where they have weaknesses.

**The for loop**

The for loop is obviously the most common type of loop there is, so this should be a quick refresher.

**const** digits = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];

**for** (**let** i = 0; i < digits.length; i++) {

console.log(digits[i]);

}

***Prints:*** *0  
1  
2  
3  
4  
5  
6  
7  
8  
9*

Really the biggest downside of a for loop is having to keep track of **the counter** and **exit condition**.

In this example, we’re using the variable i as a counter to keep track of the loop and to access values in the array. We’re also using digits.length to determine the exit condition for the loop. If you just glance at this code, it can sometimes be confusing exactly what’s happening; especially for beginners.

While for loops certainly have an advantage when looping through arrays, some data is not structured like an array, so a for loop isn’t always an option.

**The for...in loop**

The for...in loop improves upon the weaknesses of the for loop by eliminating the counting logic and exit condition.

**const** digits = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];

**for** (**const** index **in** digits) {

console.log(digits[index]);

}

***Prints:*** *0  
1  
2  
3  
4  
5  
6  
7  
8  
9*

But, you still have to deal with the issue of using an **index** to access the values of the array, and that stinks; it almost makes it more confusing than before.

Also, the for...in loop can get you into big trouble when you need to add an extra method to an array (or another object). Because for...in loops loop over all enumerable properties, this means if you add any additional properties to the array's prototype, then those properties will also appear in the loop.

Array.prototype.decimalfy = **function**() {

**for** (**let** i = 0; i < **this**.length; i++) {

**this**[i] = **this**[i].toFixed(2);

}

};

**const** digits = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];

**for** (**const** index **in** digits) {

console.log(digits[index]);

}

***Prints:*** *0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
function() {  
 for (let i = 0; i < this.length; i++) {  
  this[i] = this[i].toFixed(2);  
 }  
}*

Gross! This is why for...in loops are discouraged when looping over arrays.

***NOTE:****The****forEach loop****is another type of for loop in JavaScript. However, forEach() is actually an array method, so it can only be used exclusively with arrays. There is also no way to stop or break a forEach loop. If you need that type of behavior in your loop, you’ll have to use a basic for loop.*