Fastest loop in JavaScript

In web development, JavaScript is the new sensation. It is not just JS frameworks like NodeJS, React, Angular Vue, etc, vanilla JS also has a large fan base. Now let’s talk about modern JavaScript. Almost every programming language uses loops. The modern JS language gives you a great deal of flexibility when it comes to iterating over values.

The question is, do you know which loop or iteration fits your needs best. A variety of options are available in for loops, including for , for(reverse), for…of , foreach , for…in , and for…await . The article presents one such debate.

**Which is the fastest for loop?**

***Answer****:* **for (reverse)**

* I’m surprised that for (reverse) is the fastest of all the for loops when I tested it locally. Here’s an example. Run for a loop through an array with over one million items.
* *Please note that console.time() results are highly dependent on your system configuration. Check out the accuracy here.*
* *const million = 1000000;  
  const arr = Array(million);  
  console.time(‘⏳’);  
  for (let i = arr.length; i > 0; i — ) {} // for(reverse) :- 1.5ms  
  for (let i = 0; i < arr.length; i++) {} // for :- 1.6ms  
  arr.forEach(v => v) // foreach :- 2.1ms  
  for (const v of arr) {} // for…of :- 11.7ms  
  console.timeEnd(‘⏳’);*
* The reverse for loop and the forward for loop take almost the same amount of time. *for(reverse)* calculates a starting variable let*i = arr.length* only once, so there is a 0.1ms difference. After each increment in the forward for loop, it checks the condition*i < arr.length.* It will make no difference, ignore it.
* On the other hand, *foreach*is a method of an array prototype. Comparatively to normal *for loops*, *foreach* and *for…of* takes longer to iterate through the array.

**What loops are there, and when should you use them?**

**1. For loop (forward and reverse)**

* Everyone is probably familiar with this loop. If you need to repeat a block of code to fix counter times, you can use *for*loops.
* Traditionally, the for loop is the fastest, so you should always use them, right? Not necessarily. Performance is not the only factor. In general, *code readability* is more important, so choose the style that fits your application.

**2. forEach**

* Upon receiving an array element, this method executes a callback function for each element. Furthermore, *foreach’s* callback function accepts the current value and the index.
* *foreach*also allows you to use *this*keyword as an optional parameter within the callback function.
* const things = [‘have’, ‘fun’, ‘coding’];  
  const callbackFun = (item, idex) => {  
  console.log(`${item} — ${index}`);  
  }  
  things.foreach(callbackFun);  
  o/p:- have — 0  
  fun — 1  
  coding — 2

In JavaScript, you cannot take advantage of short-circuiting if you use *foreach*. Let me introduce you to short-circuiting if you are unfamiliar with it. When we use a logical operator in JavaScript, like *AND(&&)*, *OR(||)* we can bypass an iteration of a loop.

**3. For…of**

This for…of is standardized in ES6(ECMAScript 6). By using the for..of loop, you can iterate over an iterable object such as an array, map, set, string, etc. In addition, you can make the code more readable.

*const arr = [3, 5, 7];  
const str = ‘hello’;  
for (let i of arr) {  
console.log(i); // logs 3, 5, 7  
}  
for (let i of str) {  
console.log(i); // logs ‘h’, ‘e’, ‘l’, ‘l’, ‘o’  
}*

***Note:*** *For…of* should never be reused on generators, even if for…of ends early. The generator is turned off after exiting the loop, and trying to repeat it again produces no more results.

**4. For…in**

* The *for…in* iterates over a variable that specifies all the enumerable properties of a given object. The *for…in* statement will return the names of your user-defined properties along with the numeric indexes for every distinct property.
* For this reason, it’s better to iterate over arrays with a for loop using a numeric index. Due to the fact that the *for…in* clause iterates over user-defined properties as well as the array elements, even if you modify the array object (by adding custom properties or methods).
* *const details = {firstName: ‘john’, lastName: ‘Doe’};  
  let fullName = ‘’;  
  for (let i in details) {  
  fullName += details[i] + ‘ ‘; // fullName: john doe  
  }*

**For…in vs. for…of**

The for…of and for…in differs primarily in the elements they iterate over. With for…in loops, you iterate over an object’s properties, whereas with for…of loops, you iterate over the values of an iterable object.

*let arr= [4, 5, 6];  
for (let i in arr) {  
console.log(i); // ‘0’, ‘1’, ‘2’  
}  
for (let i of arr) {  
console.log(i); // ‘4’, ‘5’, ‘6’  
}*

**Conclusion**

* The *for*loop is fastest, but poorly readable.
* The *foreach*is fast, and iteration is controllable.
* The *for…of* takes time, but it’s sweeter.
* The *for…in* takes time, hence less convenient.

Finally, a wise piece of advice for you. Prioritize readability. It is essential to maintain code readability when developing a complex structure at that time, but you should also pay attention to performance.

Avoid adding unnecessary extras to your code as it can slow down your app.