

TypeScript 3.2 is now available. [Download \(/#download-links\)](#) our latest version today!

Documentation ▼

# Iterators and Generators

## Iterables

An object is deemed iterable if it has an implementation for the `Symbol.iterator` ([./symbols.html#symboliterator](/symbols.html#symboliterator)) property. Some built-in types like `Array`, `Map`, `Set`, `String`, `Int32Array`, `Uint32Array`, etc. have their `Symbol.iterator` property already implemented. `Symbol.iterator` function on an object is responsible for returning the list of values to iterate on.

### **for..of** statements

`for..of` loops over an iterable object, invoking the `Symbol.iterator` property on the object. Here is a simple `for..of` loop on an array:

```
let someArray = [1, "string", false];

for (let entry of someArray) {
  console.log(entry); // 1, "string", false
}
```

### **for..of** vs. **for..in** statements

Both `for..of` and `for..in` statements iterate over lists; the values iterated on are different though, `for..in` returns a list of *keys* on the object being iterated, whereas `for..of` returns a list of *values* of the numeric properties of the object being iterated.

Here is an example that demonstrates this distinction:

```
let list = [4, 5, 6];

for (let i in list) {
  console.log(i); // "0", "1", "2",
}

for (let i of list) {
  console.log(i); // "4", "5", "6"
}
```

Another distinction is that `for..in` operates on any object; it serves as a way to inspect properties on this object. `for..of` on the other hand, is mainly interested in values of iterable objects. Built-in objects like `Map` and `Set` implement `Symbol.iterator` property allowing access to stored values.

```
let pets = new Set(["Cat", "Dog", "Hamster"]);
pets["species"] = "mammals";

for (let pet in pets) {
    console.log(pet); // "species"
}

for (let pet of pets) {
    console.log(pet); // "Cat", "Dog", "Hamster"
}
```

### Code generation

#### Targeting ES5 and ES3

When targeting an ES5 or ES3, iterators are only allowed on values of `Array` type. It is an error to use `for...of` loops on non-Array values, even if these non-Array values implement the `Symbol.iterator` property.

The compiler will generate a simple `for` loop for a `for...of` loop, for instance:

```
let numbers = [1, 2, 3];
for (let num of numbers) {
    console.log(num);
}
```


will be generated as:

```
var numbers = [1, 2, 3];
for (var _i = 0; _i < numbers.length; _i++) {
    var num = numbers[_i];
    console.log(num);
}
```

#### Targeting ECMAScript 2015 and higher

When targeting an ECMAScript 2015-compliant engine, the compiler will generate `for...of` loops to target the built-in iterator implementation in the engine.

Made with ❤ in Redmond Follow @Typescriptlang (<https://twitter.com/typescriptlang>)

Privacy (<https://go.microsoft.com/fwlink/?LinkId=521839>) ©2012-2018 Microsoft  Microsoft