`NodeList` objects are collections of nodes, usually returned by properties such as `Node.childNodes` and methods such as `document.querySelectorAll()`.

> Although `NodeList` is not an `Array`, it is possible to iterate over it with `forEach()`. It can also be converted to a real `Array` using `Array.from()`.
>
> However, some older browsers have not implemented `NodeList.forEach()` nor `Array.from()`. This can be circumvented by using `Array.prototype.forEach()` — see this document's Example.

Live vs. Static NodeLists

Although they are both considered `NodeList`s, there are 2 varieties of NodeList: *live* and *static*.

Live NodeLists

In some cases, the `NodeList` is *live*, which means that changes in the DOM automatically update the collection.

For example, `Node.childNodes` is live:

```
const parent = document.getElementById('parent');
let child_nodes = parent.childNodes;
console.log(child_nodes.length); // let's assume "2"
parent.appendChild(document.createElement('div'));
console.log(child_nodes.length); // outputs "3"
```

Static NodeLists

In other cases, the `NodeList` is *static,* where any changes in the DOM does not affect the content of the collection. The ubiquitous `document.querySelectorAll()` method returns a *static* `NodeList`.

It's good to keep this distinction in mind when you choose how to iterate over the items in the `NodeList`, and whether you should cache the list's `length`.

Properties

`NodeList.length`
The number of nodes in the `NodeList`.

Methods

`NodeList.item()`
Returns an item in the list by its index, or `null` if the index is out-of-bounds.

An alternative to accessing `nodeList[i]` (which instead returns `undefined` when `i` is out-of-bounds). This is mostly useful for non-JavaScript DOM implementations.

`NodeList.entries()`
Returns an `iterator`, allowing code to go through all key/value pairs contained in the collection. (In this case, the keys are numbers starting from `0` and the values are nodes.)

`NodeList.forEach()`
Executes a provided function once per `NodeList` element, passing the element as an argument to the function.

`NodeList.keys()`
Returns an `iterator`, allowing code to go through all the keys of the key/value pairs contained in the collection. (In this case, the keys are numbers starting from `0`.)

`NodeList.values()`
Returns an `iterator` allowing code to go through all values (nodes) of the key/value pairs contained in the collection.

Example

It's possible to loop over the items in a `NodeList` using a for loop:

```
for (let i = 0; i < myNodeList.length; i++) {
  let item = myNodeList[i];
```

```
}
```

Don't use `for...in` to enumerate the items in `NodeLists`, since they will *also* enumerate its `length` and `item` properties and cause errors if your script assumes it only has to deal with `element` objects. Also, `for..in` is not guaranteed to visit the properties in any particular order.

`for...of` loops **will** loop over `NodeList` objects correctly:

```
const list = document.querySelectorAll('input[type=checkbox]');
for (let checkbox of list) {
  checkbox.checked = true;
}
```

Recent browsers also support iterator methods (`forEach()`) as well as `entries()`, `values()`, and `keys()`.

There is also an Internet Explorer-compatible way to use `Array.prototype.forEach` for iteration:

```
const list = document.querySelectorAll('input[type=checkbox]');
Array.prototype.forEach.call(list, function (checkbox) {
  checkbox.checked = true;
});
```