Iterator □ for...of □□

□□

- Iterator□□□□□□□□
- □□ Iterator □□
- □□ Iterator □□□□□
- □□□□ Iterator □□
- Iterator □□□ Generator □□
- □□□□□□ return()□ throw()
- for...of □□
  - □□
  - Set □ Map □□
  - □□□□□□□□□
  - □□□□□□□
  - □□
  - □□□□□□□□□□
- Iterator □□
  - Iterator □□□□□□□
  - Iterator □□□□□□□

Iterator□□□□□□□□

JavaScript □□□□□ "□□"□□□□□□□□□□□□ `Array` □□□□□ `Object` □□ ES6 □□□□ `Map` □ `Set` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `Map` □ `Map` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□ Iterator □□ Iterator □□□□□□□□□□□□□□□□□□□□□□□□□□□□□

Iterator □□□

[illegible] ES6 [illegible] `for...of` [illegible] Iterator [illegible] `for...of` [illegible]

Iterator [illegible]

[illegible] 1[illegible]

[illegible] 2[illegible] `next` [illegible]

[illegible] 3[illegible] `next` [illegible]

[illegible] 4[illegible] `next` [illegible]

[illegible] `next` [illegible] `value` [illegible] `done` [illegible] `value` [illegible] `done` [illegible]

[illegible] `next` [illegible]

```
var it = makeIterator(['a', 'b']);


it.next() // { value: "a", done: false }
it.next() // { value: "b", done: false }
it.next() // { value: undefined, done: true }


function makeIterator(array) {
  var nextIndex = 0;
  return {
    next: function() {
      return nextIndex < array.length ?
        {value: array[nextIndex++], done: false} :
        {value: undefined, done: true};
    }
  };
}
```

[illegible] `makeIterator` [illegible] `['a', 'b']` [illegible] `it` [illegible]

[illegible] `next` [illegible] `next` [illegible] `a` [illegible] `b` [illegible]

`next` [illegible] `value` [illegible] `done` [illegible] `value` [illegible] `done` [illegible] `next` [illegible]

[illegible] `next` [illegible]

[illegible] `done: false` [illegible] `value: undefined` [illegible] `makeIterator` [illegible]

```
function makeIterator(array) {
  var nextIndex = 0;
  return {
    next: function() {
      return nextIndex < array.length ?
        {value: array[nextIndex++]} :
        {done: true};
    }
  };
}
```

[illegible] Iterator [illegible]

```
var it = idMaker();

it.next().value // 0
it.next().value // 1
it.next().value // 2
// ...

function idMaker() {
  var index = 0;

  return {
    next: function() {
      return {value: index++, done: false};
    }
  };
}
```

[illegible] `idMaker` [illegible]

[illegible] TypeScript [illegible] Iterable[illegible] Iterator[illegible] `next` [illegible]

```
interface Iterable {
  [Symbol.iterator]() : Iterator,
}

interface Iterator {
  next(value?: any) : IterationResult,
}

interface IterationResult {
  value: any,
  done: boolean,
}
```

[illegible] Iterator [illegible]

Iterator [illegible] `for...of` [illegible] `for...of` [illegible] Iterator [illegible]

[illegible] Iterator [illegible] “[illegible]”[illegible] iterable[illegible]

ES6 [illegible] Iterator [illegible] `Symbol.iterator` [illegible] `Symbol.iterator` [illegible] “[illegible]”[illegible] iterable[illegible] `Symbol.iterator` [illegible]
[illegible] `Symbol.iterator` [illegible] `Symbol` [illegible] `iterator` [illegible] Symbol [illegible] Symbol[illegible]

```
const obj = {
  [Symbol.iterator] : function () {
    return {
      next: function () {
        return {
          value: 1,
          done: true
        };
      }
    };
  }
```

```
};
```

[illegible] `obj` [illegible] iterable[illegible] `Symbol.iterator` [illegible] `next` [illegible] `next` [illegible] `value` [illegible] `done` [illegible]

ES6 [illegible] Iterator [illegible] `for...of` [illegible] `Symbol.iterator` [illegible] `Symbol.iterator` [illegible]

[illegible] Iterator [illegible]

- Array
- Map
- Set
- String
- TypedArray
- [illegible] arguments [illegible]
- NodeList [illegible]

[illegible] `Symbol.iterator` [illegible]

```
let arr = ['a', 'b', 'c'];
let iter = arr[Symbol.iterator]();

iter.next() // { value: 'a', done: false }
iter.next() // { value: 'b', done: false }
iter.next() // { value: 'c', done: false }
iter.next() // { value: undefined, done: true }
```

[illegible] `arr` [illegible] `arr` [illegible] `Symbol.iterator` [illegible]

[illegible] Iterator [illegible] `for...of` [illegible] Iterator [illegible] `Symbol.iterator` [illegible] `for...of` [illegible]

[illegible] Object[illegible] Iterator [illegible] Map [illegible] ES5 [illegible] Map [illegible] ES6 [illegible]

□□□□□□□□□□□ `for...of` □□□□□ Iterator □□□□□□□ `Symbol.iterator` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
class RangeIterator {
  constructor(start, stop) {
    this.value = start;
    this.stop = stop;
  }

  [Symbol.iterator]() { return this; }

  next() {
    var value = this.value;
    if (value < this.stop) {
      this.value++;
      return {done: false, value: value};
    }
    return {done: true, value: undefined};
  }
}

function range(start, stop) {
  return new RangeIterator(start, stop);
}

for (var value of range(0, 3)) {
  console.log(value); // 0, 1, 2
}
```

□□□□□□□□□□ Iterator □□□□□□ `Symbol.iterator` □□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□ “□□ ”□□□□□□

```
function Obj(value) {
  this.value = value;
  this.next = null;
}

Obj.prototype[Symbol.iterator] = function() {
  var iterator = { next: next };
```

```
  var current = this;

  function next() {
    if (current) {
      var value = current.value;
      current = current.next;
      return { done: false, value: value };
    }
    return { done: true };
  }
  return iterator;
}

var one = new Obj(1);
var two = new Obj(2);
var three = new Obj(3);

one.next = two;
two.next = three;

for (var i of one){
  console.log(i); // 1, 2, 3
}
```

□□□□□□□□□□□□□□□□□□□□ `Symbol.iterator` □□□□□□□□□□□□□□□□□□ `iterator` □□□□□□□ `next` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□ Iterator □□□□□□

```
let obj = {
  data: [ 'hello', 'world' ],
  [Symbol.iterator]() {
    const self = this;
    let index = 0;
    return {
      next() {
        if (index < self.data.length) {
          return {
            value: self.data[index++],
            done: false
```

```
          };
        }
        return { value: undefined, done: true };
      }
    };
  }
};
```

[illegible] `length` [illegible] Iterator [illegible] `Symbol.iterator` [illegible] Iterator [illegible]

```
NodeList.prototype[Symbol.iterator] = Array.prototype[Symbol.iterator];
// [illegible]
NodeList.prototype[Symbol.iterator] = [][Symbol.iterator];

[...document.querySelectorAll('div')] // [illegible]
```

NodeList [illegible] `Symbol.iterator` [illegible]

[illegible] `Symbol.iterator` [illegible]

```
let iterable = {
  0: 'a',
  1: 'b',
  2: 'c',
  length: 3,
  [Symbol.iterator]: Array.prototype[Symbol.iterator]
};
for (let item of iterable) {
  console.log(item); // 'a', 'b', 'c'
}
```

[illegible] `Symbol.iterator` [illegible]

```
let iterable = {
  a: 'a',
  b: 'b',
  c: 'c',
  length: 3,
```

```
  [Symbol.iterator]: Array.prototype[Symbol.iterator]
};
for (let item of iterable) {
  console.log(item); // undefined, undefined, undefined
}
```

[illegible] `Symbol.iterator` [illegible]

```
var obj = {};

obj[Symbol.iterator] = () => 1;

[...obj] // TypeError: [] is not a function
```

[illegible] `obj` [illegible] `Symbol.iterator` [illegible]

[illegible] `for...of` [illegible] `while` [illegible]

```
var $iterator = ITERABLE[Symbol.iterator]();
var $result = $iterator.next();
while (!$result.done) {
  var x = $result.value;
  // ...
  $result = $iterator.next();
}
```

[illegible] `ITERABLE` [illegible] `$iterator` [illegible] `next` [illegible] `done` [illegible] `next` [illegible]

[illegible] Iterator [illegible]

[illegible] Iterator [illegible] `Symbol.iterator` [illegible] `for...of` [illegible]

[illegible] 1 [illegible]

[illegible] Set [illegible] `Symbol.iterator` [illegible]

```
let set = new Set().add('a').add('b').add('c');
```

```
let [x,y] = set;
// x='a'; y='b'

let [first, ...rest] = set;
// first='a'; rest=['b','c'];
```

□ 2□□□□□□

□□□□□□□ ...□□□□□□□□□ Iterator □□□

```
// □□
var str = 'hello';
[...str] //  ['h','e','l','l','o']

// □□
let arr = ['b', 'c'];
['a', ...arr, 'd']
// ['a', 'b', 'c', 'd']
```

□□□□□□□□□□□□□□□□ Iterator □□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□ Iterator
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ Iterator
□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
let arr = [...iterable];
```

□ 3□ yield*

`yield*` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
let generator = function* () {
  yield 1;
  yield* [2,3,4];
  yield 5;
};

var iterator = generator();

iterator.next() // { value: 1, done: false }
iterator.next() // { value: 2, done: false }
iterator.next() // { value: 3, done: false }
```

```
iterator.next() // { value: 4, done: false }
iterator.next() // { value: 5, done: false }
iterator.next() // { value: undefined, done: true }
```

[illegible] 4[illegible]

[illegible]

- for...of
- Array.from()
- Map(), Set(), WeakMap(), WeakSet()[illegible] `new Map([['a',1],['b',2]])` [illegible]
- Promise.all()
- Promise.race()

[illegible] Iterator [illegible]

[illegible] Iterator [illegible]

```
var someString = "hi";
typeof someString[Symbol.iterator]
// "function"

var iterator = someString[Symbol.iterator]();

iterator.next()  // { value: "h", done: false }
iterator.next()  // { value: "i", done: false }
iterator.next()  // { value: undefined, done: true }
```

[illegible] `Symbol.iterator` [illegible] next [illegible]

[illegible] `Symbol.iterator` [illegible]

```
var str = new String("hi");

[...str] // ["h", "i"]

str[Symbol.iterator] = function() {
  return {
    next: function() {
```

```
      if (this._first) {
        this._first = false;
        return { value: "bye", done: false };
      } else {
        return { done: true };
      }
    },
    _first: true
  };
};

[...str] // ["bye"]
str // "hi"
```

□□□□□□□□□ str □ `Symbol.iterator` □□□□□□□□□□□□□□□□ `...` □□□□□□□□ `bye` □□□□□□□□□ `hi` □

Iterator □□□ Generator □□

`Symbol.iterator()` □□□□□□□□□□□□□□□□□□□□□ Generator □□□

```
let myIterable = {
  [Symbol.iterator]: function* () {
    yield 1;
    yield 2;
    yield 3;
  }
};
[...myIterable] // [1, 2, 3]

// □□□□□□□□□□□□

let obj = {
  * [Symbol.iterator]() {
    yield 'hello';
    yield 'world';
  }
};
```

```
for (let x of obj) {
  console.log(x);
}
// "hello"
// "world"
```

□□□□□□ `Symbol.iterator()`□□□□□□□□□□□□□□□□□ yield □□□□□□□□□□□□□□□

□□□□□ return()□ throw()

□□□□□□□□□ `next()`□□□□□□□□ `return()`□□□ `throw()`□□□□□□□□□□□□□□□□□□□□□□□ `next()`□□□□□□□□□□ `return()`□□□ `throw()`□□□□□□□□□□□

`return()`□□□□□□□□□□□□ `for...of`□□□□□□□□□□□□□□□□□□□ `break`□□□□□□□□ `return()`□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `return()`□□□

```
function readLinesSync(file) {
  return {
    [Symbol.iterator]() {
      return {
        next() {
          return { done: false };
        },
        return() {
          file.close();
          return { done: true };
        }
      };
    },
  };
}
```

□□□□□□□□ `readLinesSync`□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `next()`□□□□□□□□ `return()`□□□□□□□□□□□□□□□□□□ `return()`□□□

```
// □□□
for (let line of readLinesSync(fileName)) {
  console.log(line);
```

```
  break;
}

// □□□
for (let line of readLinesSync(fileName)) {
  console.log(line);
  throw new Error();
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `return()` □□□□□□□□□□□□□□□□□□□□
`return()` □□□□□□□□□□□□□□□□□□

□□□ `return()` □□□□□□□□□□□□□□□ Generator □□□□□□□

`throw()` □□□□□□□□ Generator □□□□□□□□□□□□□□□□□□□□□□□□□□□□ Generator □□□□□□□

for...of □□

ES6 □□ C++□ Java□ C# □ Python □□□□□□ `for...of` □□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□ `Symbol.iterator` □□□□□□□□□□ iterator □□□□□□□□ `for...of`
□□□□□□□□□□□□□□□□ `for...of` □□□□□□□□□□□□□□□ `Symbol.iterator` □□□

`for...of` □□□□□□□□□□□□□□□□ Set □ Map □□□□□□□□□□□□□□□□□ `arguments` □□□
DOM NodeList □□□□□□□□ Generator □□□□□□□□□□

□□

□□□□□□□ `iterator` □□□□□□□□□□ `Symbol.iterator` □□□□ `for...of`
□□

```
const arr = ['red', 'green', 'blue'];

for(let v of arr) {
  console.log(v); // red green blue
}

const obj = {};
obj[Symbol.iterator] = arr[Symbol.iterator].bind(arr);
```

```
for(let v of obj) {
  console.log(v); // red green blue
}
```

[illegible] `obj` [illegible] `arr` [illegible] `Symbol.iterator` [illegible] `obj` [illegible] `for...of` [illegible] `arr` [illegible]

`for...of` [illegible] `forEach` [illegible]

```
const arr = ['red', 'green', 'blue'];

arr.forEach(function (element, index) {
  console.log(element); // red green blue
  console.log(index);   // 0 1 2
});
```

JavaScript [illegible] `for...in` [illegible] ES6 [illegible] `for...of` [illegible]

```
var arr = ['a', 'b', 'c', 'd'];

for (let a in arr) {
  console.log(a); // 0 1 2 3
}

for (let a of arr) {
  console.log(a); // a b c d
}
```

[illegible] `for...in` [illegible] `for...of` [illegible] `for...of` [illegible] `entries` [illegible] `keys` [illegible]

`for...of` [illegible] `for...in` [illegible]

```
let arr = [3, 5, 7];
arr.foo = 'hello';

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

```
for (let i of arr) {
  console.log(i); //  "3", "5", "7"
}
```

□□□□□□ `for...of` □□□□□□□□ `arr` □ `foo` □□□

Set □ Map □□

Set □ Map □□□□□□□ Iterator □□□□□□□□□ `for...of` □□□

```
var engines = new Set(["Gecko", "Trident", "Webkit", "Webkit"]);
for (var e of engines) {
  console.log(e);
}
// Gecko
// Trident
// Webkit

var es6 = new Map();
es6.set("edition", 6);
es6.set("committee", "TC39");
es6.set("standard", "ECMA-262");
for (var [name, value] of es6) {
  console.log(name + ": " + value);
}
// edition: 6
// committee: TC39
// standard: ECMA-262
```

□□□□□□□□□□□ Set □□□ Map □□ Set □□□□□□□□□□□□□□□ Map □□□□□□□□□□□□□□□□□□□□□□□□□□□□ Map □□□□□□□□□

```
let map = new Map().set('a', 1).set('b', 2);
for (let pair of map) {
  console.log(pair);
}
// ['a', 1]
```

```
// ['b', 2]

for (let [key, value] of map) {
  console.log(key + ' : ' + value);
}
// a : 1
// b : 2
```

□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ ES6 □□□□ Set□ Map □□□□□□□□□□□□□□□□□□□□□□□□□□□

- `entries()` □□□□□□□□□□□□□□□□ `[□□, □□]` □□□□□□□□□□□□□□□□□□□□□□□□□□□□ Set□□□□□□□□□□ Map □□□ Iterator □□□□□□□□□□ `entries` □□□
- `keys()` □□□□□□□□□□□□□□□□□□□□□□□□
- `values()` □□□□□□□□□□□□□□□□□□□□□□□□

□□□

```
let arr = ['a', 'b', 'c'];
for (let pair of arr.entries()) {
  console.log(pair);
}
// [0, 'a']
// [1, 'b']
// [2, 'c']
```

□□□□□□

□□□□□□□□□□□□□□□□□□ `for...of` □□□□□□□□□ DOM NodeList □□□ `arguments` □□□□□□□

```
// □□□
let str = "hello";

for (let s of str) {
  console.log(s); // h e l l o
}
```

```
// DOM NodeList□□
let paras = document.querySelectorAll("p");

for (let p of paras) {
  p.classList.add("test");
}

// arguments□□
function printArgs() {
  for (let x of arguments) {
    console.log(x);
  }
}
printArgs('a', 'b');
// 'a'
// 'b'
```

□□□□□□□□ `for...of` □□□□□□□□□□□□□□□□□ 32 □ UTF-16 □□□

```
for (let x of 'a\uD83D\uDC0A') {
  console.log(x);
}
// 'a'
// '\uD83D\uDC0A'
```

□□□□□□□□□□□□□□□ Iterator □□□□□□□□□□□□□□□□□□□ `Array.from` □□□□□□□□□

```
let arrayLike = { length: 2, 0: 'a', 1: 'b' };

// □□
for (let x of arrayLike) {
  console.log(x);
}

// □□
for (let x of Array.from(arrayLike)) {
  console.log(x);
}
```

□□

□□□□□□□□ `for...of` □□□□□□□□□□□□□□□□□□□□ Iterator □□□□□□□□□□□□□□□□□□□□ `for...in` □□□□□□□□□□□□□□□

```
let es6 = {
  edition: 6,
  committee: "TC39",
  standard: "ECMA-262"
};

for (let e in es6) {
  console.log(e);
}
// edition
// committee
// standard

for (let e of es6) {
  console.log(e);
}
// TypeError: es6[Symbol.iterator] is not a function
```

□□□□□□□□□□□□□□□□ `for...in` □□□□□□□□□□ `for...of` □□□□□□□

□□□□□□□□□□□ `Object.keys` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
for (var key of Object.keys(someObject)) {
  console.log(key + ': ' + someObject[key]);
}
```

□□□□□□□□□ Generator □□□□□□□□□□□□□

```
const obj = { a: 1, b: 2, c: 3 }

function* entries(obj) {
  for (let key of Object.keys(obj)) {
    yield [key, obj[key]];
  }
}
```

```
for (let [key, value] of entries(obj)) {
  console.log(key, '->', value);
}
// a -> 1
// b -> 2
// c -> 3
```

□□□□□□□□□

□□□□□□ JavaScript □□□□□□□□□□□□□□□□□ `for` □□□

```
for (var index = 0; index < myArray.length; index++) {
  console.log(myArray[index]);
}
```

□□□□□□□□□□□□□□□□□□□ `forEach` □□□

```
myArray.forEach(function (value) {
  console.log(value);
});
```

□□□□□□□□□□□□□□□□ `forEach` □□□ `break` □□□ `return` □□□□□□□□

`for...in` □□□□□□□□□□□□

```
for (var index in myArray) {
  console.log(myArray[index]);
}
```

`for...in` □□□□□□□□

- □□□□□□□□□□□ `for...in` □□□□□□□□□□□ “0”□ “1”□ “2”□□
- `for...in` □□□
- □□□□□□ `for...in` □□□□□□□□□□□□□□□

□□□ `for...in` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

`for...of` □□□□□□□□□□□□□□□□□□□□□□□

```
for (let value of myArray) {
  console.log(value);
```

```
}
```

- [illegible] `for...in` [illegible] `for...in` [illegible]
- [illegible] `forEach` [illegible] `break` [illegible] `continue` [illegible] `return` [illegible]
- [illegible]

[illegible] break [illegible] `for...of` [illegible]

```
for (var n of fibonacci) {
  if (n > 1000)
    break;
  console.log(n);
}
```

[illegible] 1000 [illegible] 1000[illegible] `break` [illegible] `for...of` [illegible]

Iterator [illegible]

ES2025 [illegible] Iterator [illegible]

```
class MyIterator extends Iterator {
  next() {
    // …
  }
}
```

[illegible] Iterator [illegible] `next()` [illegible]

[illegible]

Iterator [illegible]

Iterator [illegible]

- Iterator.from()
- Iterator.concat()

[illegible] 1[illegible] Iterator.from()

[illegible]

```
const it = (function* () {
  yield 1;
  yield 2;
  yield 3;
})();

const obj = {
  [Symbol.iterator]() {
    return it;
  },
};

const it2 = Iterator.from(obj);

it2 === it // true
```

□□□□□□□□ `obj` □□□□□□□□ `it` □□□ `Iterator.from(obj)` □□□□□□□□ `it` □□□□□□□□ `it2` □ `it` □□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `next()` □□□□□□□□□□ Iterator □□□□□□□□□□ `Iterator.prototype` □□□□□□□□□ `Iterator.from()` □□□□□□□□□□□□□□□□ Iterator □□□□□

```
const it = {
  current: 0,
  next() {
    return { value: this.current++, done: false };
  },
};

const obj = {
  [Symbol.iterator]() {
    return it;
  },
};

const it2 = Iterator.from(obj);

it2 === it // false
it2.next() // {value: 0, done: false}
it.next() // {value: 1, done: false}
```

□□□□□□□□ obj □□□□□□□□□□ it □□□□□□□□□□ next() □□□□□ Iterator □□□□□□□□ Iterator.from(obj) □□□ it □□□□□□□ it2 □ it □□□□□□□□ it2 □□□ it □□ Iterator □□□□□□□ it □□□□ next() □□□□□□□□□ it2.next() □ it.next() □□□□□□□□□

□□□ Iterator.from() □□□□□□□□□□□□□□□□ [System.iterator]() □□□□□□□□□□□□ Iterator □□□□□

□□□□□□□□□□□□□□□□□□ Iterator.from() □□□□□□□□□□□□□

```
// □□□
const obj1 = (function* () {
  yield 1;
  yield 2;
  yield 3;
})();

const obj2 = Iterator.from(obj1);
obj2 === obj1 // true

// □□□
const obj3 = {
  current: 0,
  next() {
    return { value: this.current++, done: false };
  },
};

const obj4 = Iterator.from(obj4);
obj3 === obj4 // false
```

□□□□□ Obj1 □ obj3 □□□□□□□□□ Iterator.from() □□□□□□□□ Obj1 □ Iterator □□□□□□□□□□□□□ Obj3 □□ Iterator □□□□□□□□□□□□□□□□□□□□□□

□ 2□ Iterator.concat()

Iterator.concat() □□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
const a1 = [1,2];
const a2 = [3,4];
const a3 = [5,6];
```

```
const it = Iterator.concat(a1, a2, a3)

it.next() // {value: 1, done: false}
it.next() // {value: 2, done: false}
it.next() // {value: 3, done: false}
```

□□□□□□ `Iterator.concat()` □ `a1` □ `a2` □ `a3` □□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□

```
const array = [1, 2, 3];
const set = new Set([4, 5, 6]);
function* gen() {
  yield 7;
  yield 8;
  yield 9;
}

const it = Iterator.concat(array, set, gen());
[...it] // [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

□□□□□□ `Iterator.concat()` □□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□

```
const obj = {
  next() {
    return { done: true };
  },
};

const it = Iterator.concat(obj); // □□
```

□□□□□□ `obj` □□□□□□□□□□□□□□□□ `Iterator.concat()` □□□□□□□□ `Iterator.from()` □□□□□□□□□□□□□□□

```
const obj = {
  next() {
    return { done: true };
  },
};
```

```
const it = Iterator.concat(Iterator.from(Obj)); // [illegible]
```

Iterator [illegible]

Iterator [illegible]
[illegible]

```
- Iterator.prototype.every()
- Iterator.prototype.filter()
- Iterator.prototype.find()
- Iterator.prototype.flatMap()
- Iterator.prototype.forEach()
- Iterator.prototype.includes()
- Iterator.prototype.join()
- Iterator.prototype.map()
- Iterator.prototype.reduce()
- Iterator.prototype.some()
```

[illegible]

```
const result = Iterator.from(array)
  .map(x => x * 2)
  .filter(x => x > 10)
  .take(3)
  .toArray();
```

[illegible]

```
const arr = ['a', '', 'b', '', 'c', '', 'd', '', 'e'];

arr.values() // creates an iterator
  .filter(x => x.length > 0)
  .drop(1)
  .take(3)
  .map(x => `=${x}=`)
  .toArray()
// ['=b=', '=c=', '=d=']
```

□□□□□□ arr □□□□□□□□ values() □□□□□□□□□□□□□□□□□□□□ for...of □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□ Iterator □□□□□□□□

- Iterator.prototype.drop(limit)□□□□□□□□□□□□□□□ limit □□□□
- Iterator.prototype.take(limit)□□□□□□□□□□□□□□□ limit □□□□
- Iterator.prototype.toArray()□□□□□□□□□□□□□□□□

□□□ #2
□ □□□ □□□ 2020-04-16 10:29:52 UTC
□ □□□ □□□ 2026-09-24 17:10:01 UTC