Iterators & Generators - Exercises

1. Custom iterator

Write a simple iterator for an object that returns each property value.

```
const obj = {
    a: 'apple',
    b: 'banana',
    c: 'cherry'
};
obj[Symbol.iterator] = function() { ... };

for(const value of obj) {
    console.log(value);
}
```

Solution

```
obj[Symbol.iterator] = function() {
    const keys = Object.keys(this);
    let index = 0;
    return {
        next: () => {
            if(index < keys.length) {
                return { value: this[keys[index++]], done: false };
            } else {
                return { done: true };
            }
        }
    };
};
```

2. Fibonacci Generator

Create a generator function that produces the Fibonacci sequence:

```
function* fibonacci() {
  ...
}
```

```
const fib = fibonacci();

console.log(fib.next().value);  // 1

console.log(fib.next().value);  // 1

console.log(fib.next().value);  // 2
```

3. Create custom iterator

For a given array, create an iterator that returns only the odd values.

Hint: Use Symbol.iterator and check for odd values inside the next method.

Input	Output

const arr = [1,2,3,4,5,6,7,8,9,10];
const oddValues = new OddIterator(arr);
for (const value of oddValues){
    console.log(value);
}
	1
	3
	5
	7
	9

4. Range Generator

Write a generator function that takes two arguments, start and end, and generates numbers between them.

Sample Usage	Output

const rangeGen = createRange(2, 6);
console.log(rangeGen.next().value);
console.log(rangeGen.next().value);
	2
	3

5. Infinite Sequence

Develop a generator that keeps yielding an increasing number on each call, starting from 1.

Hint: This is an infinite generator. It will never set done to true.

Input	Output

let seqGen = new SequenceGenerator();
console.log(seqGen.next());
console.log(seqGen.next());
	1
	2
	3
	4

console.log(seqGen.next()); console.log(seqGen.next());	

6. Reverse Iterator

Create an iterator for an array that returns values in reverse order.

Input	Output
[1,2,3]	3, 2, 1
['a','b','c']	c, b, a
[5,4,3,2,1]	1, 2, 3, 4, 5

7. Prime Number Generator

Write a generator that yields only prime numbers.

Output: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47

8. Combination Generator

Create a generator function that takes an array and generates all possible combinations of its elements.

Input	Output
[1,2]	1, 2, 1,2
['a','b','c']	a, b, c, a,b, a,c, b,c, a,b,c
[5,6]	5, 6, 5,6

9. Skipping Iterator

Create an iterator for an array that skips every other value.

Input	Output
[1,2,3,4]	1, 3
['a','b','c','d']	a, c
[5,6,7,8,9]	5, 7, 9

10. Array Flatten Generator

Write a generator that takes a nested array and flattens it.

Input	Output
[[1,2],[3,4]]	1, 2, 3, 4
['a',['b','c'], ['d','e']]	a, b, c, d, e
[[5,6],7,[8,9]]	5, 6, 7, 8, 9

11. Char Code Generator

For a given string, write a generator that yields the char code for each character.

Input	Output
"ab"	97, 98
"cd"	99, 100
"ef"	101, 102

12. Power of Two Generator

Create a generator that yields numbers which are powers of 2.

Output: 1, 2, 4, 8, 16, 32, 64, 128, 256

13. Filtered Iterator

Create an iterator that only returns values that pass a given test function.

Input: [1,2,3,4,5] with test: x => x % 2 == 0

Output: 2, 4

Input: ['apple', 'banana', 'cherry'] with test: word => word.length > 5

Output: banana, cherry

Input: [5,6,7,8,9,10] with test: x => x > 7

Output: 8, 9, 10

14. Chained Generators

Combine multiple generators using yield*.

Input: generators producing 1, 2 and 3, 4

Output: 1, 2, 3, 4

Input: generators producing a, b and c, d

Output: a, b, c, d

Input: generators producing 5, 6 and 7, 8

Output: 5, 6, 7, 8

15. Queue Iterator

Implement an iterator for a queue data structure.

Input: queue with elements 1,2,3

Output: 1, 2, 3

Input: queue with elements a,b,c

Output: a, b, c

Input: queue with elements 5,6,7

Output: 5, 6, 7

16. Stack Generator

Implement a generator for a stack data structure.

Input: stack with elements 1,2,3 (3 being the top)

Output: 3, 2, 1

Input: stack with elements a,b,c (c being the top)

Output: c, b, a

Input: stack with elements 5,6,7 (7 being the top)

Output: 7, 6, 5

17. Slice Iterator

Create an iterator that returns a slice of an array.

Input: array [1,2,3,4,5] with slice indices 1,3

Output: 2, 3, 4

Input: array ['apple', 'banana', 'cherry', 'date'] with slice indices 0,2

Output: apple, banana, cherry

Input: array [5,6,7,8,9,10] with slice indices 4,5

Output: 9, 10

18. Repeat Generator

Write a generator that repeats a given value a specified number of times.

Input: value 2 repeated 3 times

Output: 2, 2, 2

Input: value “a” repeated 4 times

Output: a, a, a, a

Input: value 7 repeated 2 times

Output: 7, 7

19. Pattern Generator

Create a generator that yields a repetitive pattern of values.

Input: pattern [1,2,3] repeated 2 times

Output: 1, 2, 3, 1, 2, 3

Input: pattern ['a','b'] repeated 3 times

Output: a, b, a, b, a, b

Input: pattern [5,6] repeated 2 times

Output: 5, 6, 5, 6

20. Decimal to Binary Converter

Use a generator to convert decimal numbers to binary.

Input	Output
5	101
8	1000
15	1111

21. Map Iterator

Create an iterator for a map that returns both the key and value.

Input: map with pairs {1:'a', 2:'b', 3:'c'}

Output: 1-a, 2-b, 3-c

Input: map with pairs {a: 1, b: 2, c: 3}

Output: a-1, b-2, c-3

Input: map with pairs {x:'apple', y:'banana', z:'cherry'}

Output: x-apple, y-banana, z-cherry

22. Set Comprehension

Use generators to perform set comprehension operations.

Input: set {1,2,3,4,5} with comprehension x => x*2

Output: {2, 4, 6, 8, 10}

Input: set {a,b,c} with comprehension x => x.toUpperCase()

Output: {A, B, C}

Input: set {5,6,7,8,9} with comprehension x => x-4

Output: {1, 2, 3, 4, 5}

23. Value Transformation Generator

Create a generator that transforms input values based on a function.

Input: values [1,2,3] with function x => x*2

Output: 2, 4, 6

Input: values ['apple', 'banana'] with function word => word.length

Output: 5, 6

Input: values [5,6,7] with function x => x*x

Output: 25, 36, 49

24. Pagination Iterator

Create an iterator that simulates paginated results from a large dataset.

Input: dataset [1,2,3,4,5,6,7,8,9,10] with page size 3

Output: page 1: 1,2,3, page 2: 4,5,6, page 3: 7,8,9, page 4: 10

Input: dataset ['a','b','c','d','e','f'] with page size 2

Output: page 1: a,b, page 2: c,d, page 3: e,f