# Generators

comparison between generators and regular functions, use case of a generator

A **generator** is a special function that returns an iterator. There are some differences between generator functions and regular functions:

- There is an `*` after the `function` keyword.
- Generator functions create iterators.
- We use the `yield` keyword in the created iterator function. By writing `yield v`, the iterator returns `{ value: v, done: false }` as a value.
- We can also use the return keyword to end the iteration. Similar to iterators, the returned value won't be enumerated by a data consumer.
- The yielded result is the next value of the iteration process. Execution of the generator function is stopped at the point of yielding. Once a data consumer asks for another value, execution of the generator function is resumed by executing the statement after the last yield.

Consider the following example:

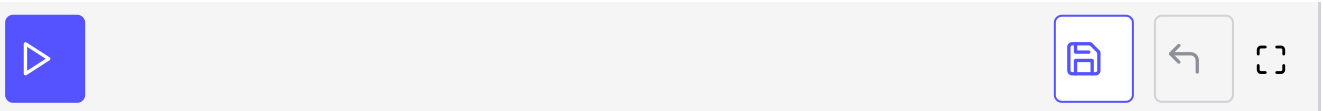
```
function *getLampIterator() {
    yield 'red';
    yield 'green';
    return 'lastValue';
    // implicit: return undefined;
}

let lampIterator = getLampIterator();

console.log( lampIterator.next() );
//> {value: "red", done: false}

console.log( lampIterator.next() );
//> {value: "green", done: false}

console.log( lampIterator.next() );
//> {value: "lastValue", done: true}
```



If the return value were missing, the function would return `{value: undefined, done: true}`.

> Use generators to define custom iterables to avoid using the well-known symbol `Symbol.iterator`.

In the next lesson, we'll discuss how generators can return iterables using various methods and operators.