Generators in JavaScript

Generator Basics

Maaike van Putten
Lead Trainer & Software Developer

@brightboost | www.brightboost.nl

Generators in JavaScript

Version Check

Version Check

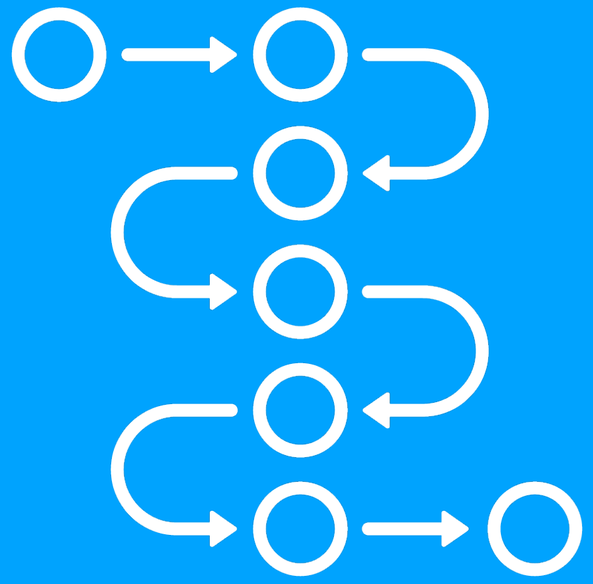
This course was created by using:

- ES14 (ECMAScript 2023)

Version Check

This course is 100% applicable to:

- ES7 (ECMAScript 2016)



Generators

Special functions that can be paused and resumed.

Overview

What a generator is

What they are used for

Relevance of generators for async programming

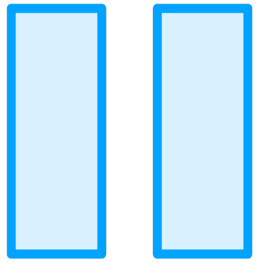
Write and use your own generator functions

- function*
- yield
- next()
- yield*

If you have a solid understanding of generator functions already, you might want to skip this module.

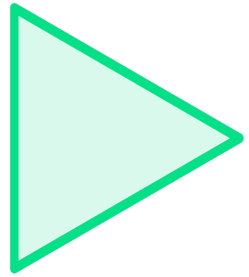
Understanding generator functions

Generator Functions



Pause

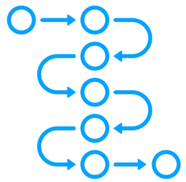
Generators can be paused mid-execution



Resume

Execution can be resumed from the point they were paused

Generators and Yield



Generators don’t run in the usual way

Special object to run it step-by-step

Pause is indicated by the yield keyword

Yield can be used comparable to the return keyword

Use Cases for Generators

Retrieving a series of values

You might be thinking, why not use a list instead?

Use Cases for Generators

Retrieving a series of values

Generators are optimized for memory

Because generators don’t need to hold all the values in memory, only the latest generated one.

Use Cases for Generators

Retrieving a series of values

Generators are optimized for memory

Can be used for streaming data

Generating values

Lazy Evaluation

Nothing is generated, unless it’s actively requested.
This is great for performance optimization because
it prevents unnecessary generation.

Use Cases for Generators

Retrieving a series of values

Generators are optimized for memory

Can be used for streaming data

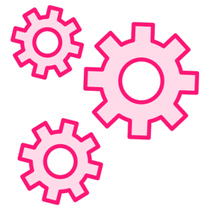
Generating values

Asynchronous programming

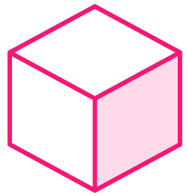
Low-Level Construct

You probably won’t use them daily. However, they are a fundamental building block of JavaScript and understanding them advances your JavaScript knowledge.

Generators



Tool for tools



Building block that’s used in tools

Reading and understanding source code

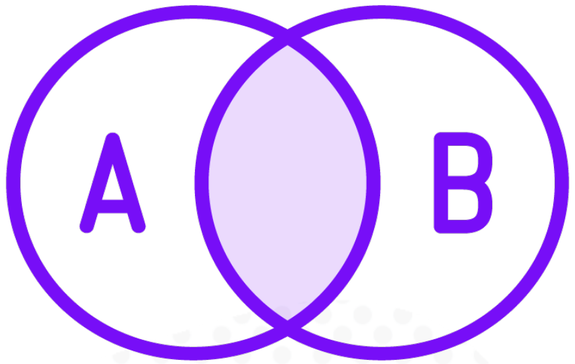
Generators Often Have an Alternative

Multiple options

For example, regular for loop and for of loop

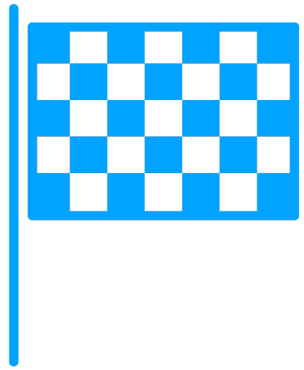
Sometimes, best practice

Sometimes, matter of preference



Creating generator functions

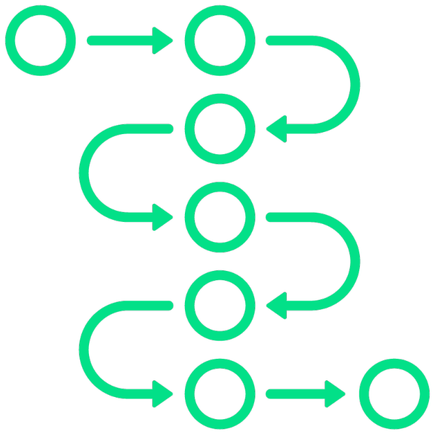
Regular Functions and Generator Functions



Regular Functions

Run-to-completion

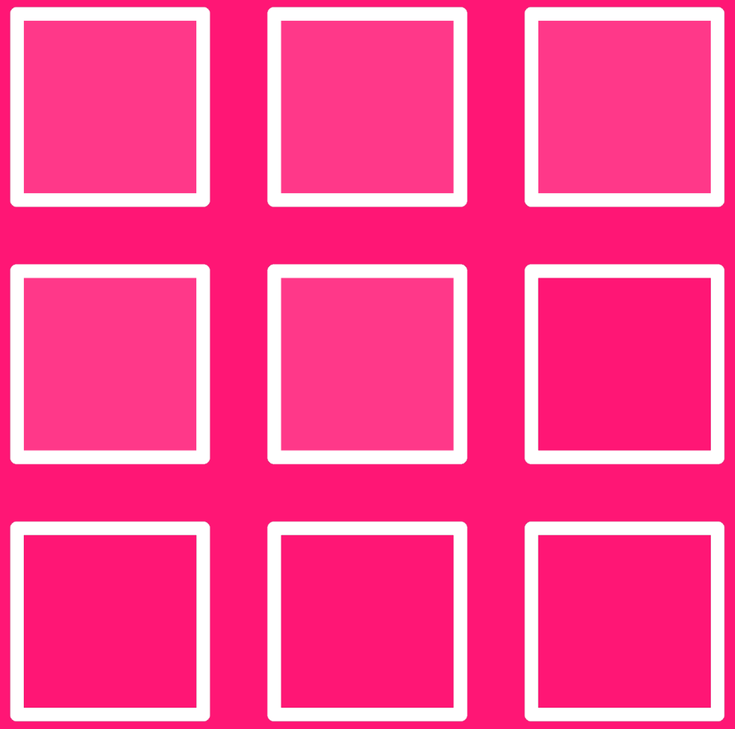
Once triggered, they run until done



Generator Functions

Pause and resume

They run until they are paused



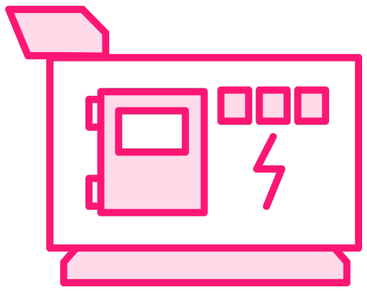
Using Generators

Obtain the generator object and use it to execute the generator function and get the values.

Using generators with next()

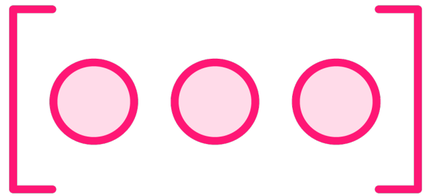
Executing a Generator Function

In order to execute, we need to have the following three elements:



Generator

Generator object



Execute the generator

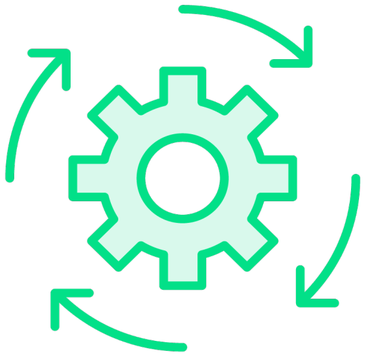
Generator is also an iterator

- Use it with the for...of loop
- Combine it with the spread operator

Why use a generator here instead of anything else?

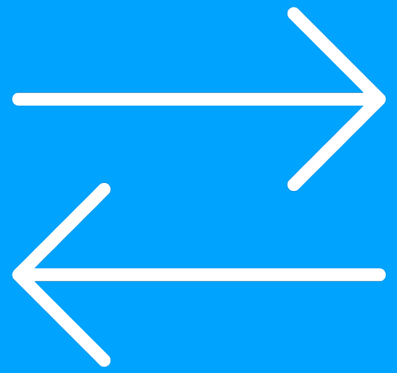
Benefits of Generators

Lazy evaluation



Stateful iteration

Control flow



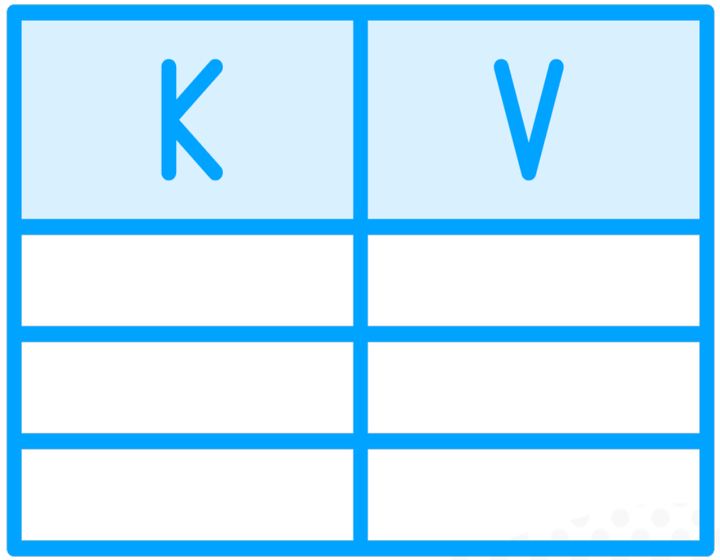
Two Way Street

We can send data back to the generator.

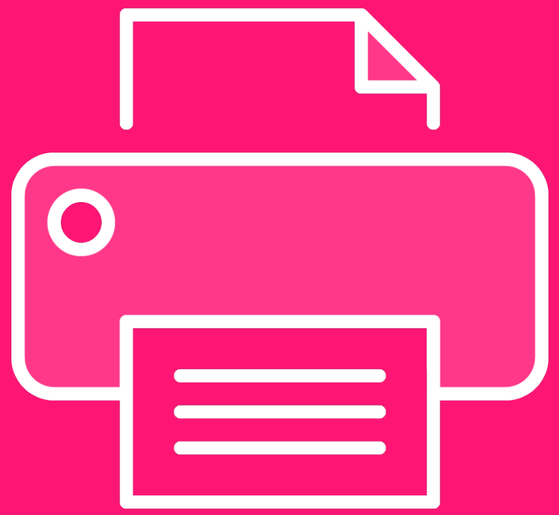
Done property

Next yields an object

Holds a value and done property



The object with value and done



Let’s Print the Objects

We’re going to inspect the object that next() is sending back to see what is happening.

We can return a value when we're done with the return keyword.

Cancel generators

Cancel the Execution of a Generator

We can cancel it with the return method and by having the generator throw an error.

Use Cases for Canceling Execution

Resource management

Long-running tasks

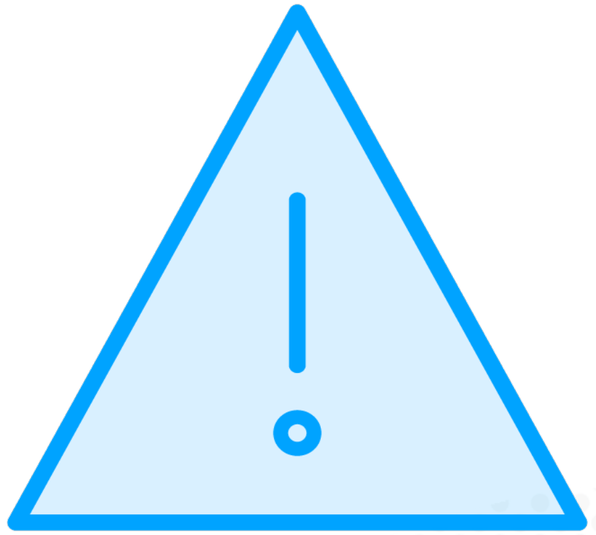
User interactions

Error handling

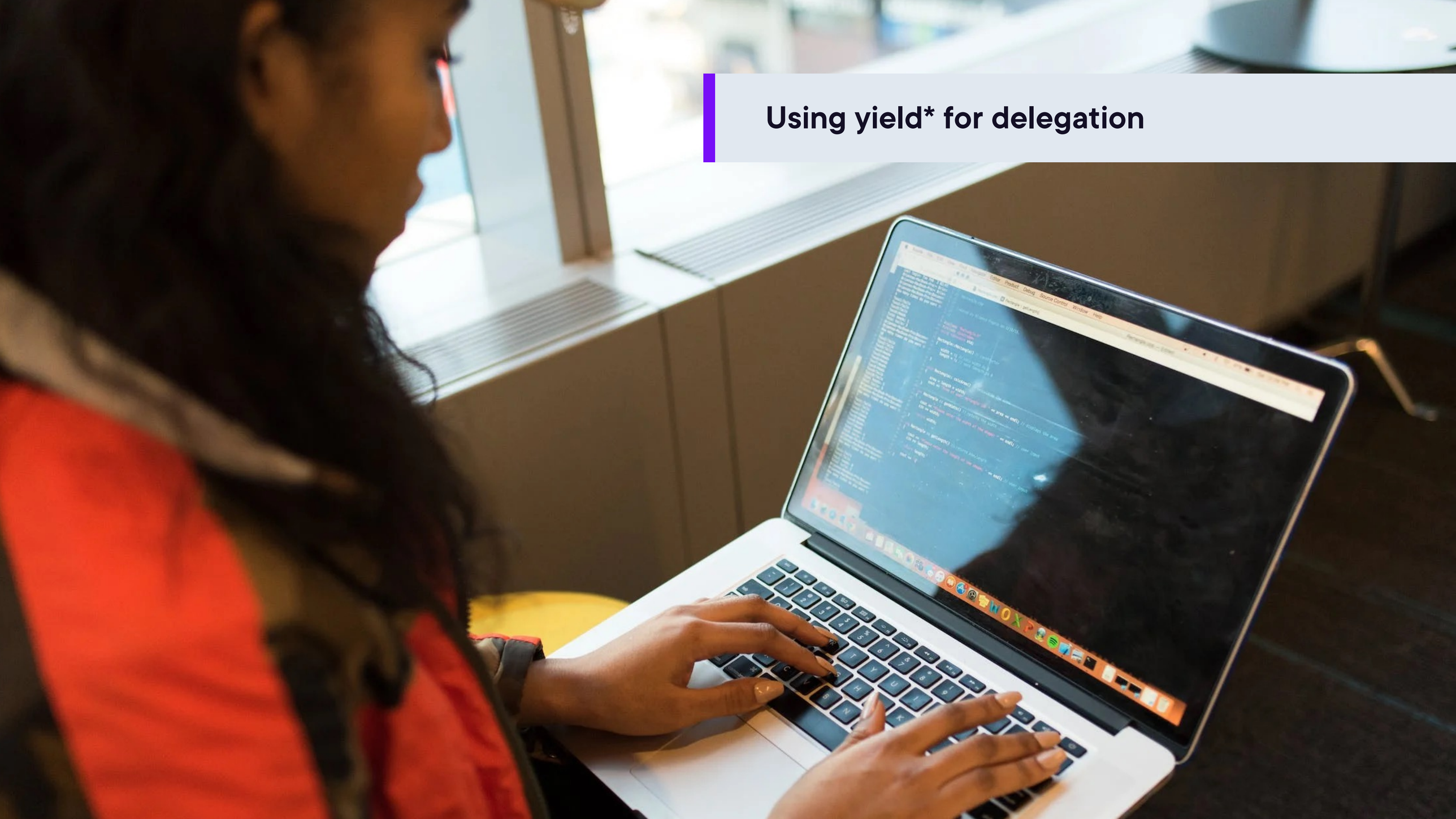
Async flow control

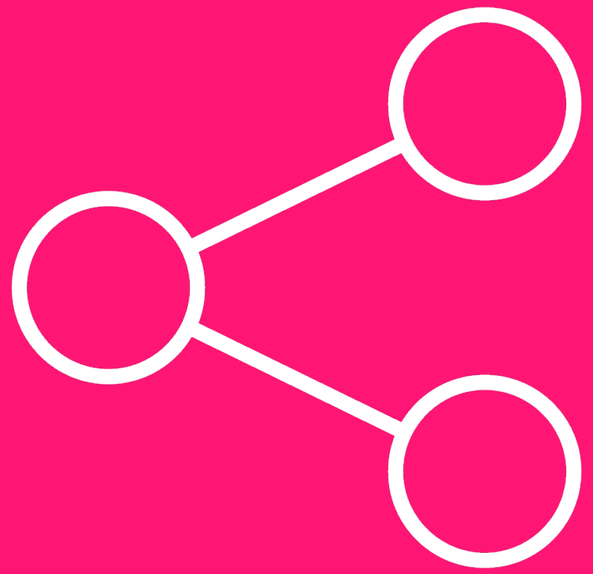
No actual use cases so far

Demonstrating the mechanisms



Using yield* for delegation

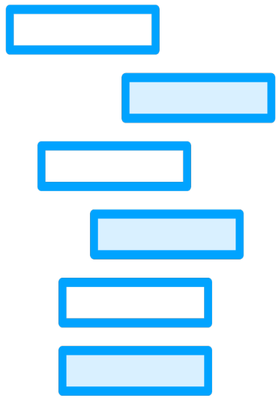




Delegating Generators with yield*

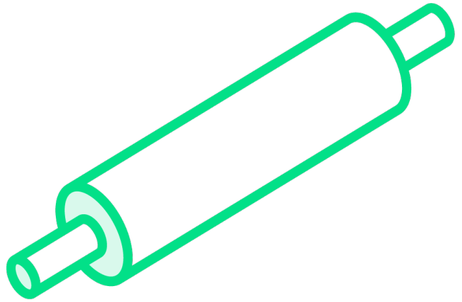
We'll see a generator function that will yield another generator.

Why Use Delegation?



Composition

Combine multiple smaller generators into one bigger complex generator



Iterable flattening

Work with nested iterables while having readable code

Demo

We'll need multiple generators

Create a generator that is yielding the other generators

Module summary

Summary

What a generator is

What they are used for

Relevance of generators for async programming

Write and use your own generator functions

- function*
- yield
- next()
- yield*

Up Next:

Working with Generator Patterns