

# Best Practices and Production Concerns



**Maaike van Putten**

Lead Trainer & Software Developer

@brightboost | [www.brightboost.nl](http://www.brightboost.nl)

# Overview



**Debug generator functions**

**Evaluate the performance of generators**

**Performance optimization**

**Security and generators**



A professional woman with long brown hair tied back, wearing black-rimmed glasses and a white blazer over a light-colored top, is seated at a desk and looking down at her laptop screen. She is positioned in front of a large window with a view of green trees. A purple vertical bar is located on the left side of the slide.

## Debugging generators



# Debugging Generators

Similar to debugging other JS code

Common bugs

Challenging due to paused and resumed execution

Logging and debugging tools





# Internal State

**Can lead to bugs when not managed or fully understood correctly**

**Careful with external state that is crucial for the inner state of the generator**



## Demo



**Debugging a problem with the state of the generator**





## Errors in the Generator

This can lead to unexpected outcomes that are challenging to debug.



## Demo



**Debugging an error thrown in the generator function**



**Errors occurring in a generator can be tricky.  
Let's implement the best practice of adding a try catch.**



# Demo



## Handling the error inside the generator



A professional woman with brown hair tied back, wearing glasses and a green blazer over a white shirt, is seated at a desk in an office. She is looking down at her laptop, which is open on the desk in front of her. The office has large windows in the background, letting in natural light. There are some plants on the windowsill. A purple vertical bar is positioned on the left side of the slide.

## Performance profiling



A photograph of a woman wearing a maroon hijab and a white blouse with black grid patterns. She is sitting at a desk, looking down at a silver laptop. Her hands are visible on the keyboard. Behind her is a window with light-colored curtains and a potted plant on the windowsill.

# Profiling the Performance of a Generator

## Measuring:

- Execution time
- Memory usage
- Identify bottlenecks



## Demo



**Every situation of profiling is different**

**Walk through the different steps:**

- Measure time with `console.time()`
- Using DevTools for a detailed analysis





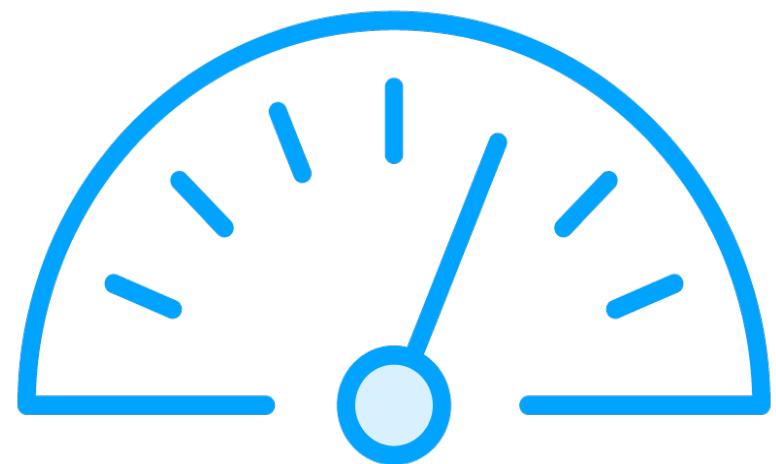
**Check the code where the memory usage spikes**

**These are the bottlenecks**

**For generators this could be the generator itself, but also the code processing the yielded values**



# Optimize Code



**Identified bottlenecks**

**Rewrite certain parts of the generator**

**More efficient algorithms**

**Optimize processing of yielded data**

**Try without generator if possible**

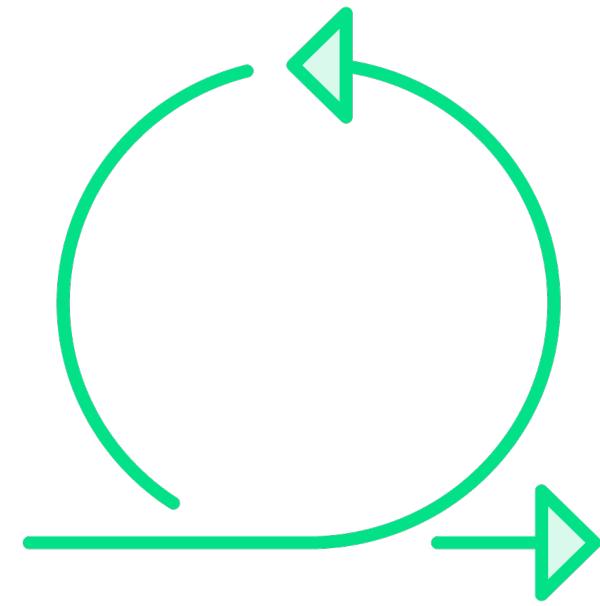
**Re-run the profiler to see the difference**



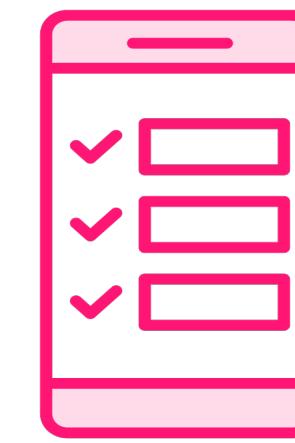
# Iterative Process



Profile

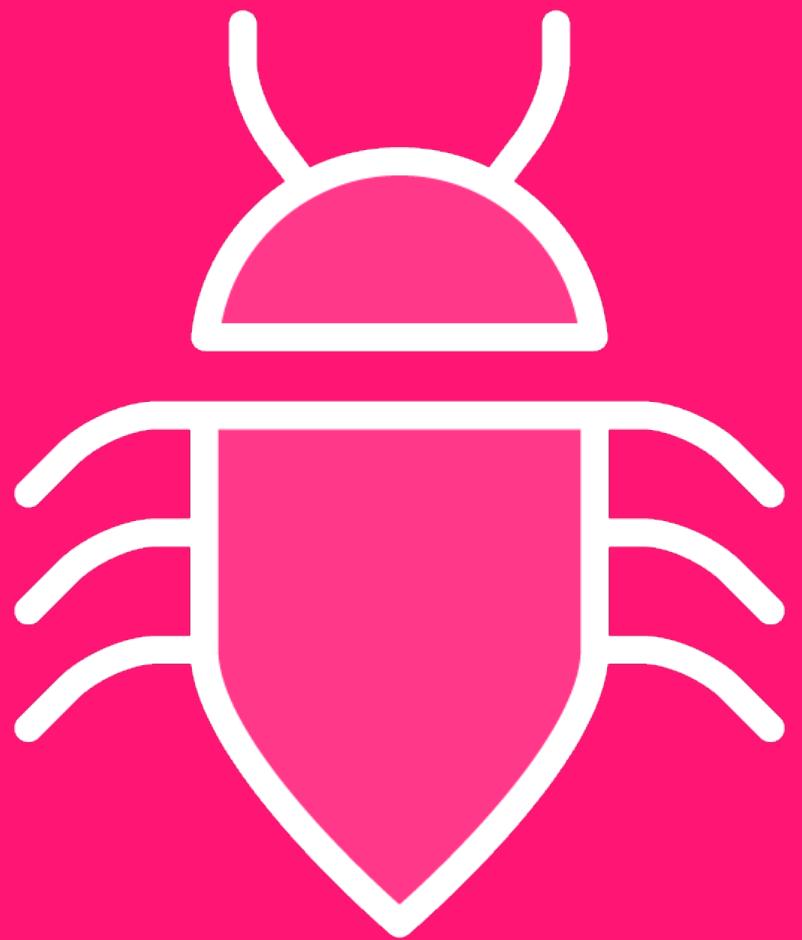


Optimize



Retest





## Speed Is Just One Part

We're also looking at memory efficiency.



## Performance optimization tips







## Optimization techniques

- Memory management
- Lazy evaluation
- Asynchronous operations

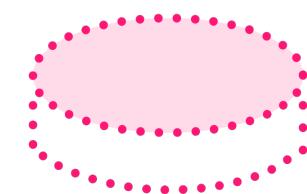
## Pitfalls

- Additional complexity
- Overusing generators
- Infinite loops

## General best practices



# Memory Management



**Generators are typically memory-efficient**



**Yield smaller chunks**



**Find balance: making API calls and database calls is heavy on performance too**



**Because the dataset can be  
processed in chunks.**



# Memory Management



**Generators are typically memory-efficient**



**Yield smaller chunks**



**Find balance: making API calls and database calls is heavy on performance too**





# Lazy Evaluation

**Strong point of generators**

**Reduce initial load time**

**Reduce memory footprint**

**Avoid unnecessary computations**





# Complex Asynchronous Situation

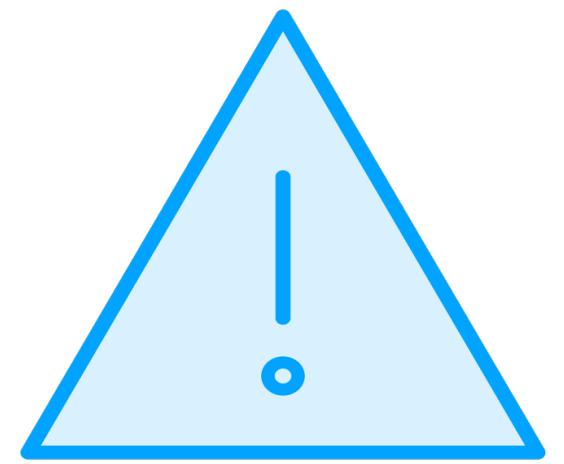
**Handle enormous amounts of data async**

**More readable than complex nested callbacks  
and promise chains**

**More control than just async/await**



# Generator Pitfalls



**Don't overuse  
generators**



# Use the Best Tool for the Job



**Whenever a simpler solution is available, such as:**

- Loop
- Array
- Async/await

**Generators are great for their use cases but add complexity**

**Whenever generators yield the result of an API or database call they generate overhead**



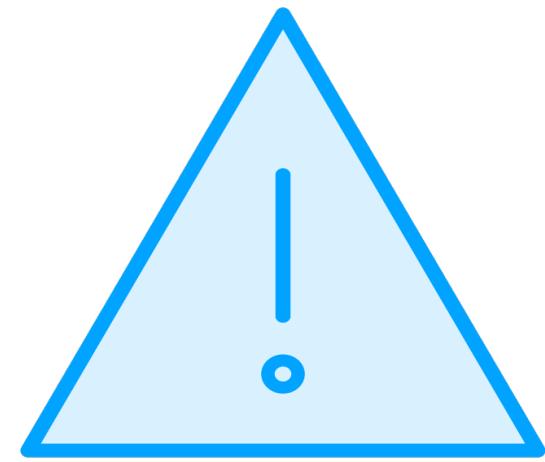


## Using Generators or Alternatives

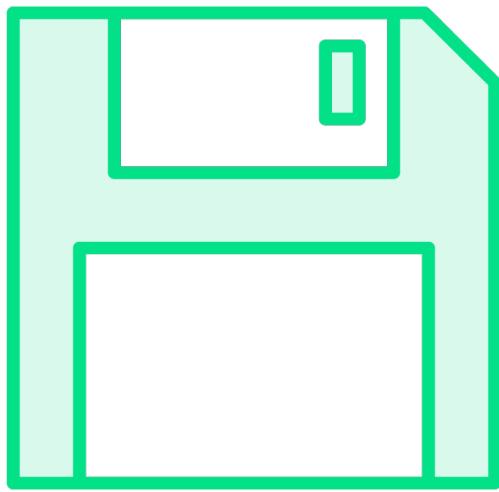
Assess the requirement, consider the possible solutions and choose the cleanest and highest performing solution.



# Generator Pitfalls



**Don't overuse  
generators**



**Don't use generators  
for small datasets**



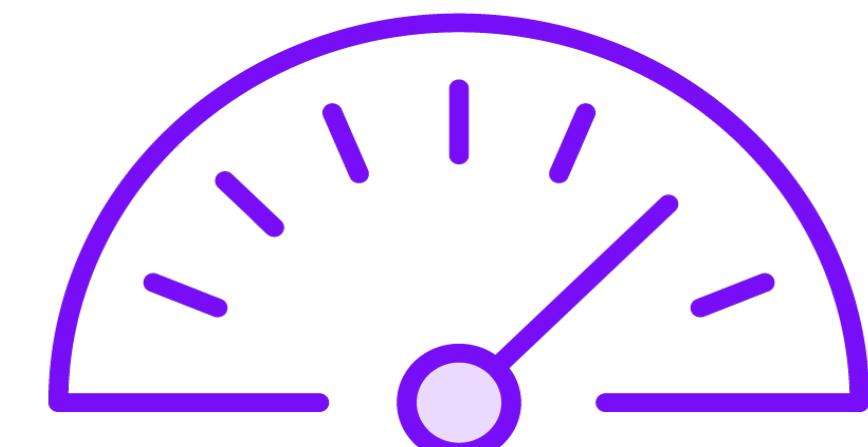
# Generators and Overhead

**Generators don't add overhead**

**Often they yield the result of a DB or API call**

**Those calls are heavy on the performance**

**It's only considered overhead when it's a  
small dataset**



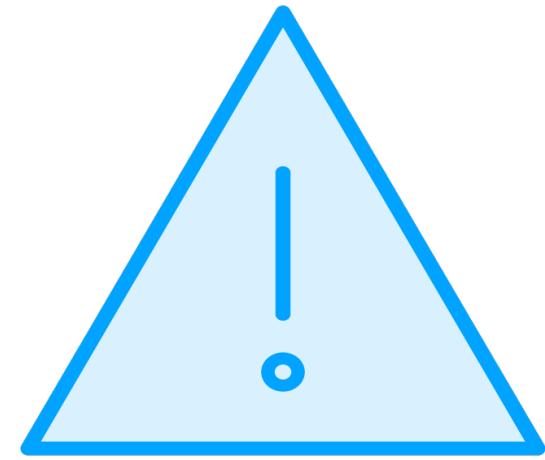


## Choosing the Best Solution

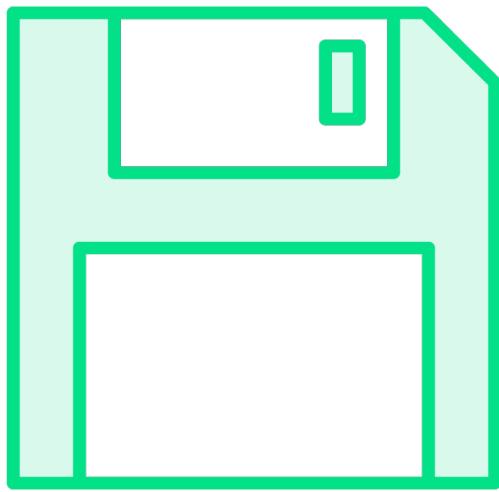
**Profile your code and try it with and without a generator to see what it does to performance.**



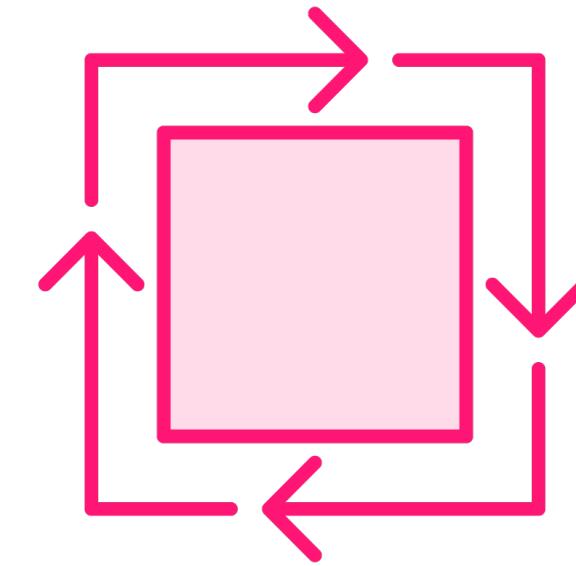
# Generator Pitfalls



**Don't overuse  
generators**



**Don't use generators  
for small datasets**



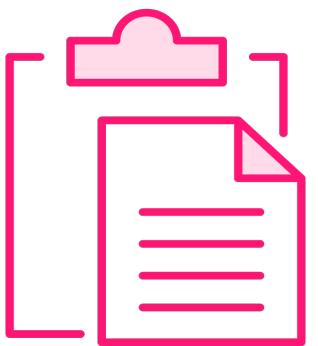
**Be cautious of infinite  
loops**



# Best Practices



**Testing**



**Documentation**



**Keep It Simple (Stupid)**



A group of diverse professionals are gathered around a wooden conference table in a modern office. In the foreground, a woman with curly hair and glasses is smiling and looking at a laptop screen. Next to her, a man in a tan blazer is also looking at the laptop. To the right, a woman in a pink blazer is leaning in, and another person's hands are visible on the right side. On the far left, a man wearing glasses is partially visible. The background shows a corkboard with various sticky notes pinned to it.

## Security and generators





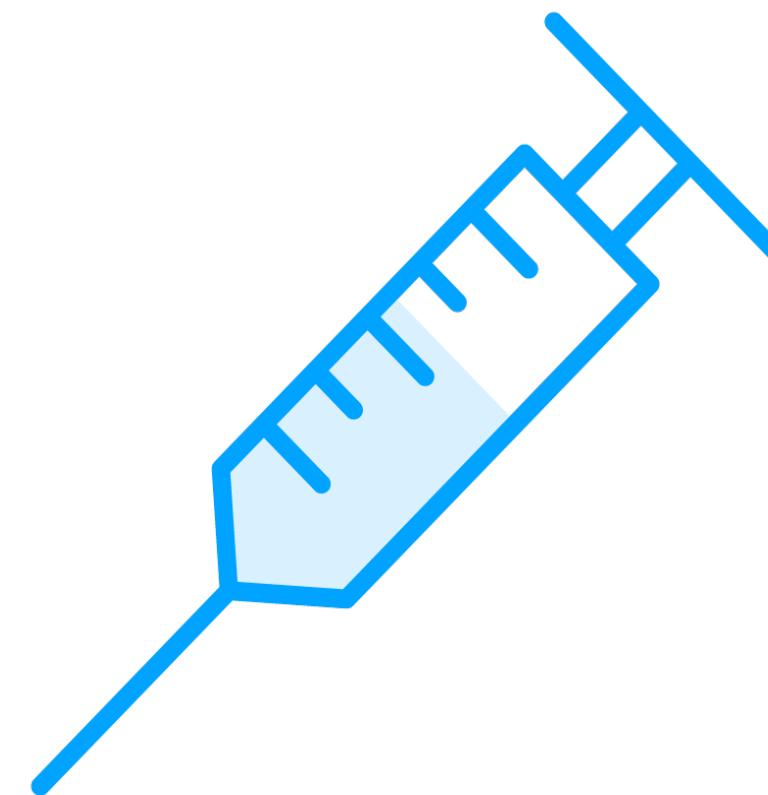
# Security and Generators

Generators are not especially dangerous

Considerations and potential issues can arise for  
any code construct



# Injection Attacks in Generator Functions



**Not generator specific**

**Injection attacks possible when generators interact with the database or external APIs**

**Validate and sanitize user inputs and use prepared statements**



# Resource Exhaustion

**Risk in the context of large datasets,  
intensive computations or long running  
processes**

**Could be exploited in DoS attacks**

**Implement limits and handling resource limits**

**Monitor and manage resource usage**





# Side Effects and State Management

**Generators maintain state**

**State could include sensitive data**

**Carefully manage state to avoid unintended side effects**





## Generators Are Not Inherently Insecure

Like any code construct, security best practices must be applied to reduce risks.



## Course wrap up





# Summary



**Debug generator functions**

**Evaluate the performance of generators**

**Performance optimization**

**Security and generators**



A close-up photograph of a person's hands writing in a notebook. The person is wearing a silver ring on their left hand and a black leather watch on their right wrist. The notebook has lined pages with handwritten text. In the background, there is a green mug on a saucer and some colorful cards or photos on a wooden surface.

Reflect on what you've learned



## Be Like the Generators in Learning

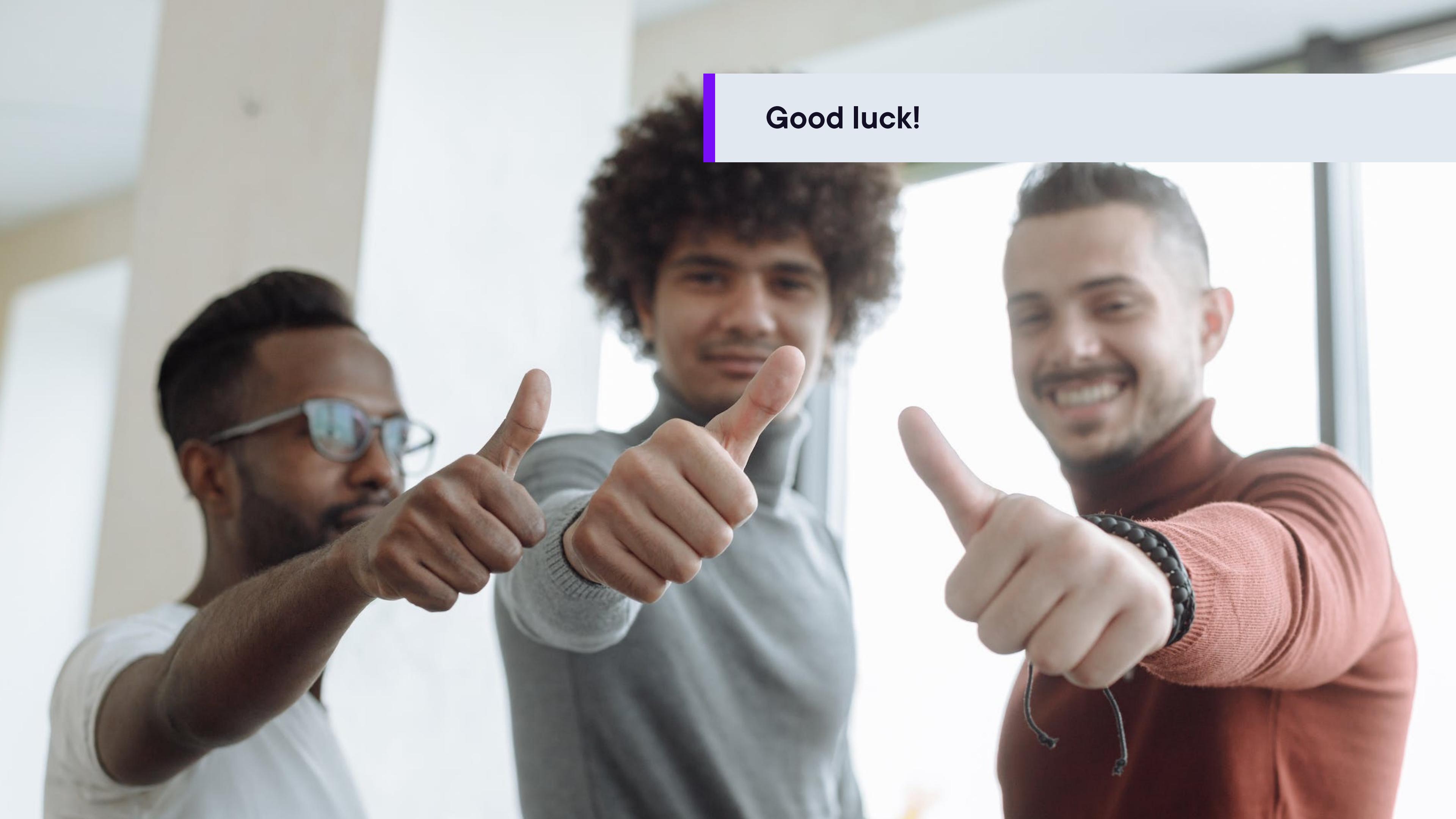
**Don't process everything all at once**

**Learning should be done in manageable chunks**

**Iterate through each concept**

**Gradually build your skillset**



A photograph of three young men of diverse ethnicities standing side-by-side, all giving a thumbs-up gesture towards the camera. They are all smiling. The man on the left is wearing glasses and a white t-shirt. The man in the center has curly hair and is wearing a grey hoodie. The man on the right has short hair and is wearing a reddish-brown turtleneck sweater. The background is a bright, possibly indoor space.

Good luck!