# Iterables, Iterators and Generators

## Overview

In this lab you'll implement a generator function to perform custom iteration on a collection of objects. The generator function will be a method in a class, for added excitement.

This lab is a bit different from the previous ones. Rather than implementing a full-blown web application, you'll just focus on writing your code in a script file and then run it directly via Babel. This will allow you to focus on the concepts and syntax of generators, which is quite enough to be getting on with.

## Source folders

* C:/JsDeepDive/Labs/Student/08-IterGen
* C:/JsDeepDive/Labs/Solutions/08-IterGen

## Roadmap

There are 3 exercises in this lab, of which the last exercise is "if time permits". Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

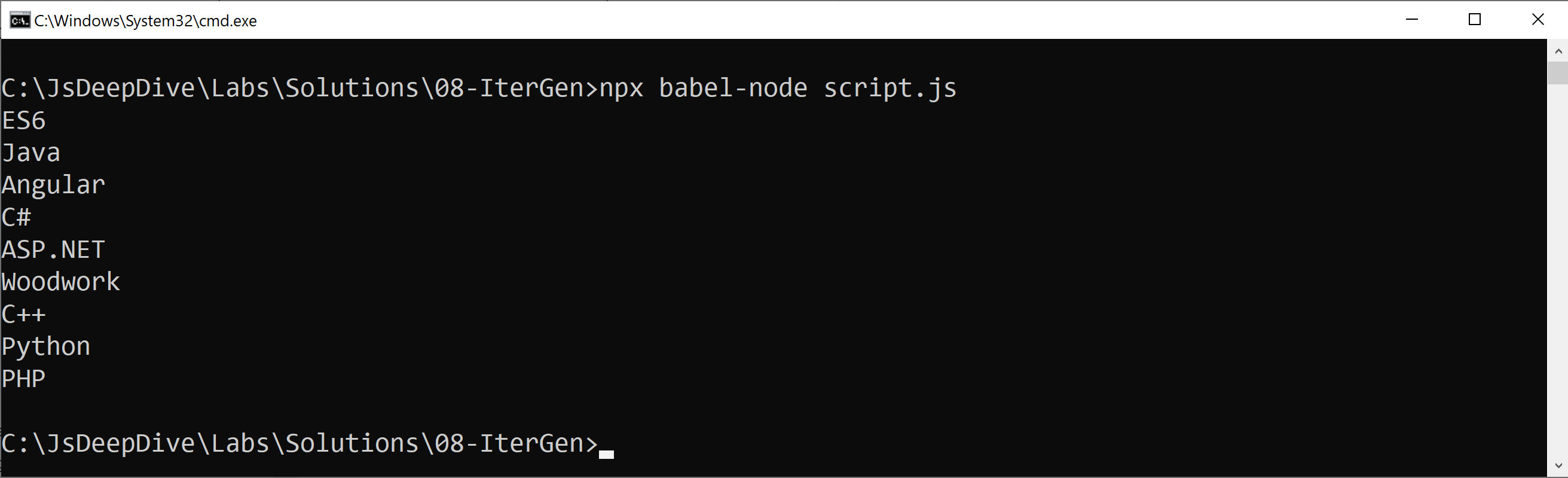
1. Familiarization with the solution application
2. Implementing the student code
3. (If Time Permits) Additional suggestions

## Exercise 1: Familiarization with the solution application

Open a Command Prompt window and go to the **solution** folderfor this lab. Note there's only a single file, script.js. Run this script file via Babel as follows:

npx babel-node script.js

The code has a generator function that performs custom iteration on a supplied collection of objects. Specifically in this example, the generator function iterates through a collection of Person objects and yields all the skills for all the people:



## Exercise 2: Implementing the student code

Now go to the **student** folderand openscript.js in a text editor. Note the following points in the code:

* The Person class is just a sample class. The constructor sets the name, skills, and contacts for a person.
* In the client code, we create an array of Person objects. The names, skills, and contacts are actually space-separated strings.
* The client code also shows how to use the FieldExpander class – this is the class you'll implement in this lab…

The FieldExpander class is very general purpose. The constructor takes 3 parameters:

* An iterable collection of items (e.g. an array here).
* The name of the property that you want to pluck out of each object, e.g. 'skills' here. The assumption is that the property will be a space-separated string of sub-values (for example).
* An optional separator (the default separator should be a space).

The FieldExpander class has a generator method named values(), which yields all the unique sub-values for the specified property on the specified collection.

Define the FieldExpander class, in accordance with the usage we've just explained in the client code. Refer to the screenshot at the top of the page for a reminder of how it should look. When you’re ready, run your script via Babel and verify the results are correct.

## Exercise 3 (If time permits): Additional suggestions

1. Try iterating over other properties in the Person objects.
2. Define another class in a similar vein, and try iterating over its properties.
3. Experiment with other types of collection than array, e.g. Map and Set.