## Iterating and the Asynchronous Loop

This lesson shows how to iterate with asynchronous loops.

## WE'LL COVER THE FOLLOWING

- ^
- Iteration and the TypeScript Generator
- Before TypeScript 3.2: Working with the asynciterator

## Iteration and the TypeScript Generator #

TypeScript generators are advanced and very modern concept. They use the function\* syntax which is the standard function keyword followed by an asterisk. The star syntax indicates that a function returns a generator object. A function that returns a generator object can return multiple times. This function allows the use of the yield keyword, which returns a value without returning the function. It allows for potential infinite iteration, while still being able to consume the value outside the function. The function returns a generator object which has a .next() function. The next function returns a value of type IteratorResult. The function returns when the iteration is over with a done property from IteratorResult. The done property is of type Boolean and is used to tell a while or a for loop to stop.

## Before TypeScript 3.2: Working with the **asynciterator**

Prior to **TypeScript 3.2**, it was not straightforward to set up TypeScript to work with async iterators. In both cases, here are the steps in which the difference will be discussed.

The **first step** is to change **tsconfig.json** to use the library **esnext.asynciterator**.

The **second step** is to redefine the asynciterator with the any hack. However,

The example needs to stimulate an action that takes time, which will be a **delay function** that returning a promise after a specific number of milliseconds. You also need a method to generate something every time the delay expires. In this example, you will generate a random set of strings.

The **third step** is to create a **function\*** which yields a return value. The method can return a single value or an array. When yielding an array, a start must follow the **yield** keyword. To demonstrate both methods, the loop yields a single set of characters before the delay and two sets after by using the array syntax.

Finally, the function\* must be invoked. You can refer to the following example at **step 4**, where function\* is looping. The loop uses the keyword await to wait for the next yielded value.

```
// Step 1 (Older version of TypeScript only)
// (<any>Symbol).asyncIterator = Symbol.asyncIterator || Symbol.for("Symbol.asyncIterator");
// Step 2
function delay(ms: number): Promise<void> {
    return new Promise<void>(resolve => {
        setTimeout(resolve, ms);
    });
function getRandomSetChars(): string {
    const random = 1 + Math.floor(Math.random() * 5);
    let wordString = "";
    for (let i = 0; i < random; i++) {
        const letter = 97 + Math.floor(Math.random() * 26);
        wordString += String.fromCharCode(letter);
    return wordString;
// Step 3
async function* getRandomSetsChars(): AsyncIterableIterator<string> {
   for (let i = 0; i < 10; i++) {
        yield getRandomSetChars(); // return a random set of char
        await delay(200); // wait
        yield* [getRandomSetChars(), getRandomSetChars()]; // return two random sets of char
// Step 4
async function addWordsAsynchronously() {
    for await (const x of getRandomSetsChars()) {
        console.log("Iterator loop:" + x);
addWordsAsynchronously();
```