



- ✓ 3. Symbols
- ✓ 4. Iteration & Iterable Protocols
- ✓ 5. Sets
- ✓ 6. Modifying Sets
- ✓ 7. Working with Sets
- ✓ 8. Sets & Iterators
- ✓ 9. Quiz: Using Sets
- ✓ 10. WeakSets
- ✓ 11. Quiz: Working With WeakSets
- ✓ 12. Maps
- ✓ 13. Creating & Modifying Maps
- ✓ 14. Working with Maps
- ✓ 15. Looping Through Maps
- ✓ 16. WeakMaps
- ✓ 17. Promises Intro
- ✓ 18. Promises
- ✓ 19. More Promises
- ✓ 20. Proxies Intro
- ✓ 21. Proxies
- ✓ 22. Proxies vs. ES5 Getter/Setter
- ✓ 23. Proxies Recap
- ✓ 24. Generators
- 25. Generators & Iterators
- 26. Sending Data into/out of a Gene...
- 27. Lesson 3 Summary

Whenever a function is invoked, the JavaScript engine starts at the top of the function and runs every line of code until it gets to the bottom. There's no way to stop the execution of the function in the middle and pick up again at some later point. This "**run-to-completion**" is the way it's always been:

```
function getEmployee() {  
  console.log('the function has started');  
  
  const names = ['Amanda', 'Diego', 'Farrin', 'James', 'Kagure', 'Kavita', 'Orit', 'Richard'];  
  
  for (const name of names) {  
    console.log(name);  
  }  
  
  console.log('the function has ended');  
}  
  
getEmployee();
```

Running the code above produces the following output on the console:

```
the function has started  
Amanda  
Diego  
Farrin  
James  
Kagure  
Kavita  
Orit  
Richard  
the function has ended
```

But what if you want to print out the first 3 employee names then stop for a bit, then, at some later point, you want to continue where you left off and print out more employee names. With a regular function, you can't do this since there's no way to "pause" a function in the middle of its execution.

Pausable Functions

If we *do* want to be able to pause a function mid-execution, then we'll need a new type of function available to us in ES6 - generator functions! Let's look at one:

```
function* getEmployee() {  
  console.log('the function has started');  
  
  const names = ['Amanda', 'Diego', 'Farrin', 'James', 'Kagure', 'Kavita', 'Orit', 'Richard'];  
  
  for (const name of names) {  
    console.log( name );  
  }  
  
  console.log('the function has ended');  
}
```

Notice the asterisk (i.e. `*`) right after the `function` keyword? That asterisk indicates that this function is actually a generator!

Now check out what happens when we try running this function:

```
getEmployee();  
  
// this is the response I get in Chrome:  
getEmployee {[[GeneratorStatus]]: "suspended", [[GeneratorReceiver]]: Window}
```



- ✓ 3. Symbols
- ✓ 4. Iteration & Iterable Protocols
- ✓ 5. Sets
- ✓ 6. Modifying Sets
- ✓ 7. Working with Sets
- ✓ 8. Sets & Iterators
- ✓ 9. Quiz: Using Sets
- ✓ 10. WeakSets
- ✓ 11. Quiz: Working With WeakSets
- ✓ 12. Maps
- ✓ 13. Creating & Modifying Maps
- ✓ 14. Working with Maps
- ✓ 15. Looping Through Maps
- ✓ 16. WeakMaps
- ✓ 17. Promises Intro
- ✓ 18. Promises
- ✓ 19. More Promises
- ✓ 20. Proxies Intro
- ✓ 21. Proxies
- ✓ 22. Proxies vs. ES5 Getter/Setter
- ✓ 23. Proxies Recap
- ✓ 24. Generators
- 25. Generators & Iterators
- 26. Sending Data into/out of a Gene...
- 27. Lesson 3 Summary

QUIZ QUESTION

Which of the following are valid generators? Pay attention to the placement of the asterisk.

If you're not sure, try running them in your browser's console.

☐ `function* names() { /* ... */ }`

☐ `function * names() { /* ... */ }`

☐ `function *names() { /* ... */ }`

SUBMIT

NEXT