JavaScript yield

Summary: in this tutorial, you will learn about the JavaScript `yield` keyword and how to use it in generator functions.

Introduction to the JavaScript yield keyword

The `yield` keyword allows you to pause and resume a generator function (`function*`).

The following shows the syntax of the `yield` keyword:

```
[variable_name] = yield [expression];
```

In this syntax:

- The `expression` specifies the value to return from a generator function via the iteration protocol. If you omit the `expression` , the `yield` returns `undefined` .
- The `variable_name` stores the optional value passed to the `next()` method of the iterator object.

JavaScript yield examples

Let's take some examples of using the `yield` keyword.

A) Returning a value

The following trivial example illustrates how to use the `yield` keyword to return a value from a generator function:

```
function* foo() {
    yield 1;
    yield 2;
```

```
    yield 3;
}

let f = foo();

console.log(f.next());
```

Output:

```
{ value: 1, done: false }
```

As you can see the value that follows the `yield` is added to the `value` property of the return object when the `next()` is called:

```
yield 1;
```

B) Returning undefined

This example illustrates how to use the `yield` keyword to return `undefined`:

```
function* bar() {
    yield;
}

let b = bar();
console.log(b.next());
```

Output:

```
{ value: undefined, done: false }
```

C) Passing a value to the next() method

In the following example, the `yield` keyword is an expression that evaluates the argument passed to the `next()` method:

```
function* generate() {
    let result = yield;
    console.log(`result is ${result}`);
}

let g = generate();
console.log(g.next());

console.log(g.next(1000));
```

The first call `g.next()` returns the following object:

```
{ value: undefined, done: false }
```

The second call g.next() carries the following tasks:

- Evaluate `yield` to 1000.
- Assign `result` the value of `yield`, which is `1000`.
- Output the message and return the object

Output:

```
result is 1000
{ value: undefined, done: true }
```

D) Using yield in an array

The following example uses the `yield` keyword as elements of an array:

```
function* baz() {
    let arr = [yield, yield];
    console.log(arr);
}

var z = baz();
```

```
console.log(z.next());
console.log(z.next(1));
console.log(z.next(2));
```

The first call `z.next()` sets the first element of the `arr` array to 1 and returns the following object:

```
{ value: undefined, done: false }
```

The second call `z.next()` sets the second of the `arr` array to 2 and returns the following object:

```
{ value: undefined, done: false }
```

The third call `z.next()` shows the contents of the `arr` array and returns the following object:

```
[ 1, 2 ]
{ value: undefined, done: true }
```

E) Using yield to return an array

The following generator function uses the `yield` keyword to return an array:

```
function* yieldArray() {
    yield 1;
    yield [ 20, 30, 40 ];
}

let y = yieldArray();

console.log(y.next());
console.log(y.next());
console.log(y.next());
```

The first call `y.next()` returns the following object:

```
{ value: 1, done: false }
```

The second call `y.next()` returns the following object:

```
{ value: [ 20, 30, 40 ], done: false }
```

In this case, `yield` sets the array `[ 20, 30, 40 ]` as the value of the `value` property of the return object.

The third call `y.next()` returns the following object:

```
{ value: undefined, done: true }
```

F) Using the yield to return individual elements of an array

See the following generator function:

```
function* yieldArrayElements() {
    yield 1;
    yield* [ 20, 30, 40 ];
}

let a = yieldArrayElements();

console.log(a.next()); // { value: 1, done: false }
console.log(a.next()); // { value: 20, done: false }
console.log(a.next()); // { value: 30, done: false }
console.log(a.next()); // { value: 40, done: false }
```

In this example, `yield*` is the new syntax. The `yield*` expression is used to delegate to another iterable object or generator.

As a result, the following expression returns the individual elements of the array `[20, 30, 40]`:

```
yield* [20, 30, 40];
```

In this tutorial, you have learned about the JavaScript `yield` keyword and how to use it in function generators.