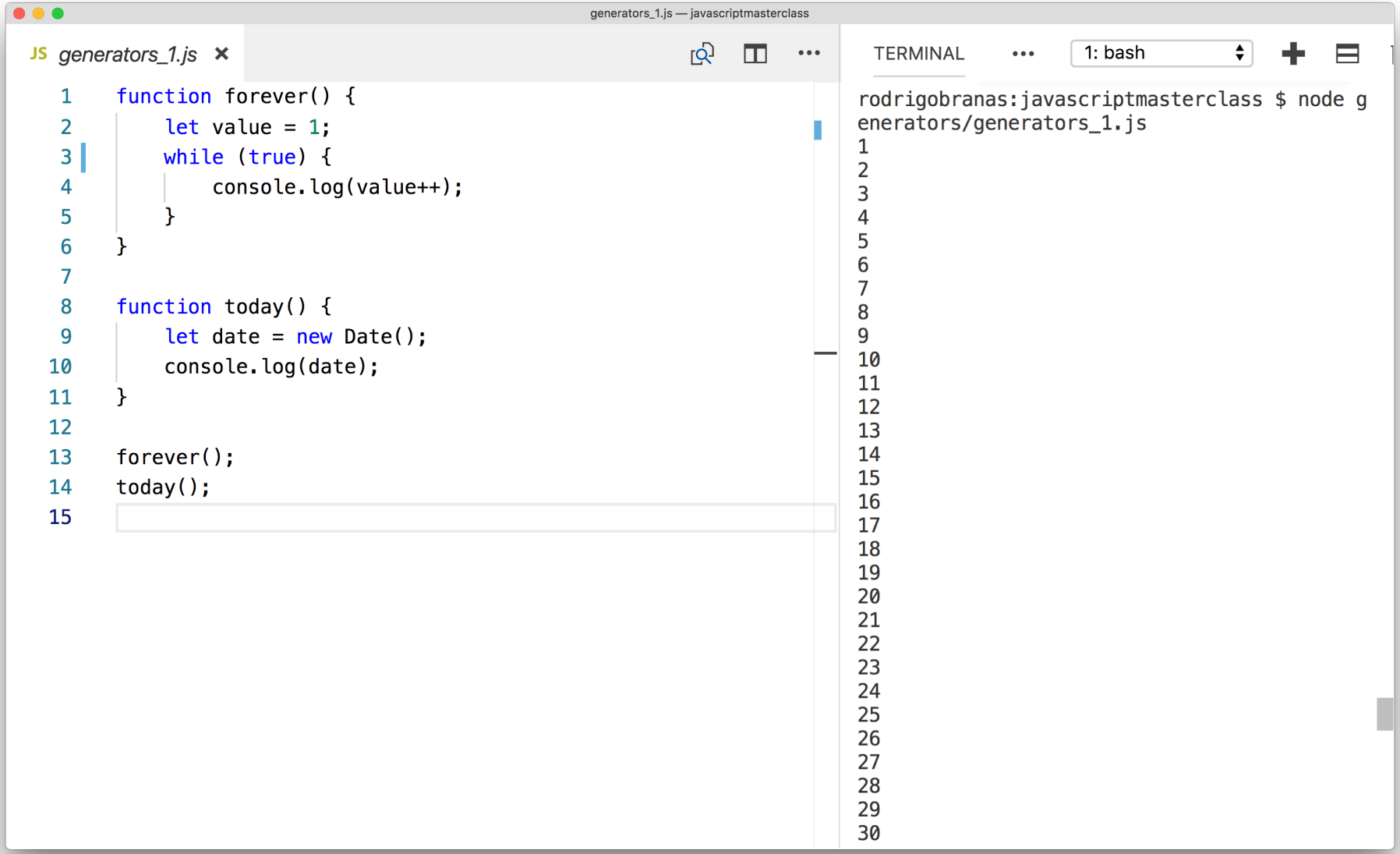
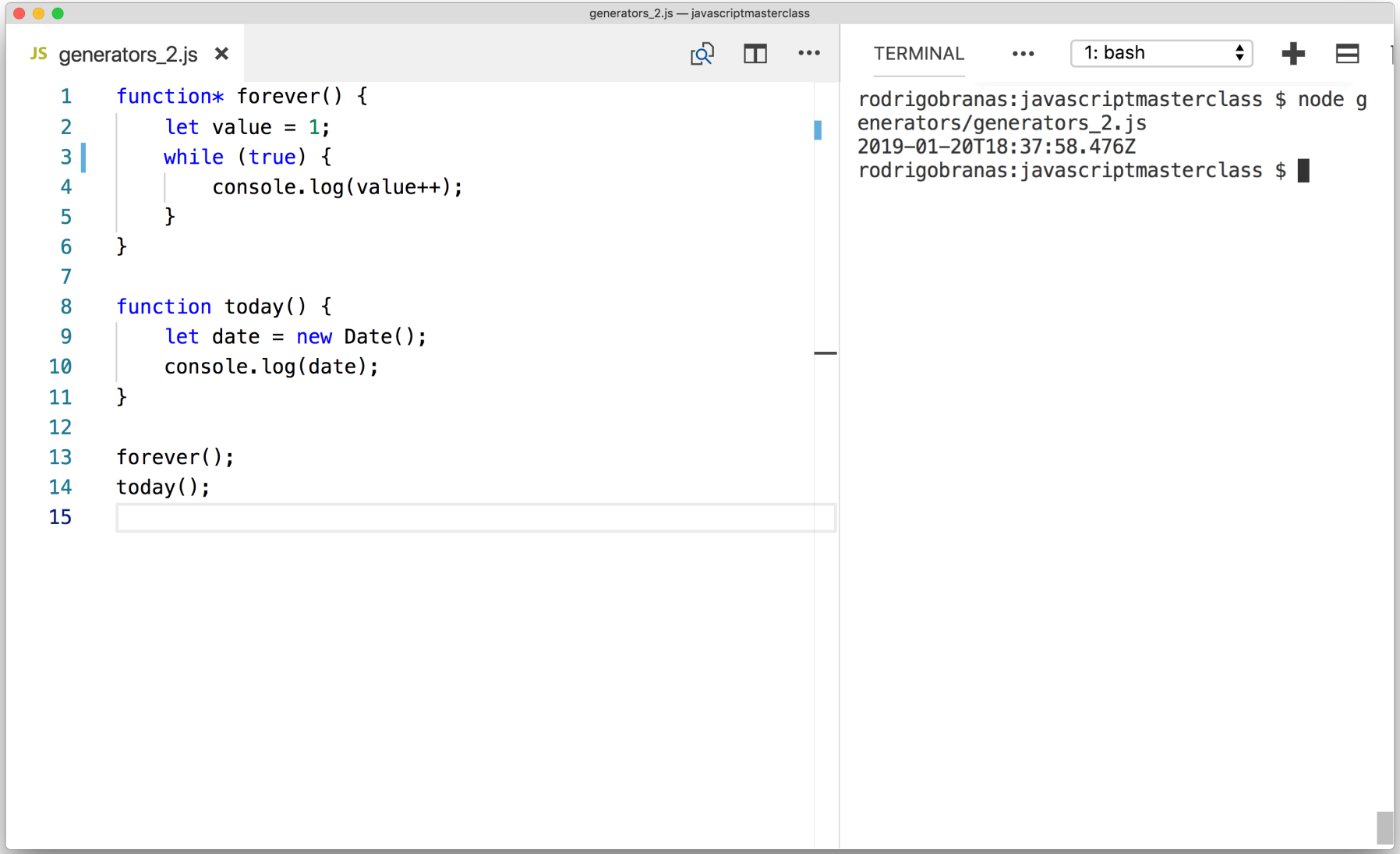
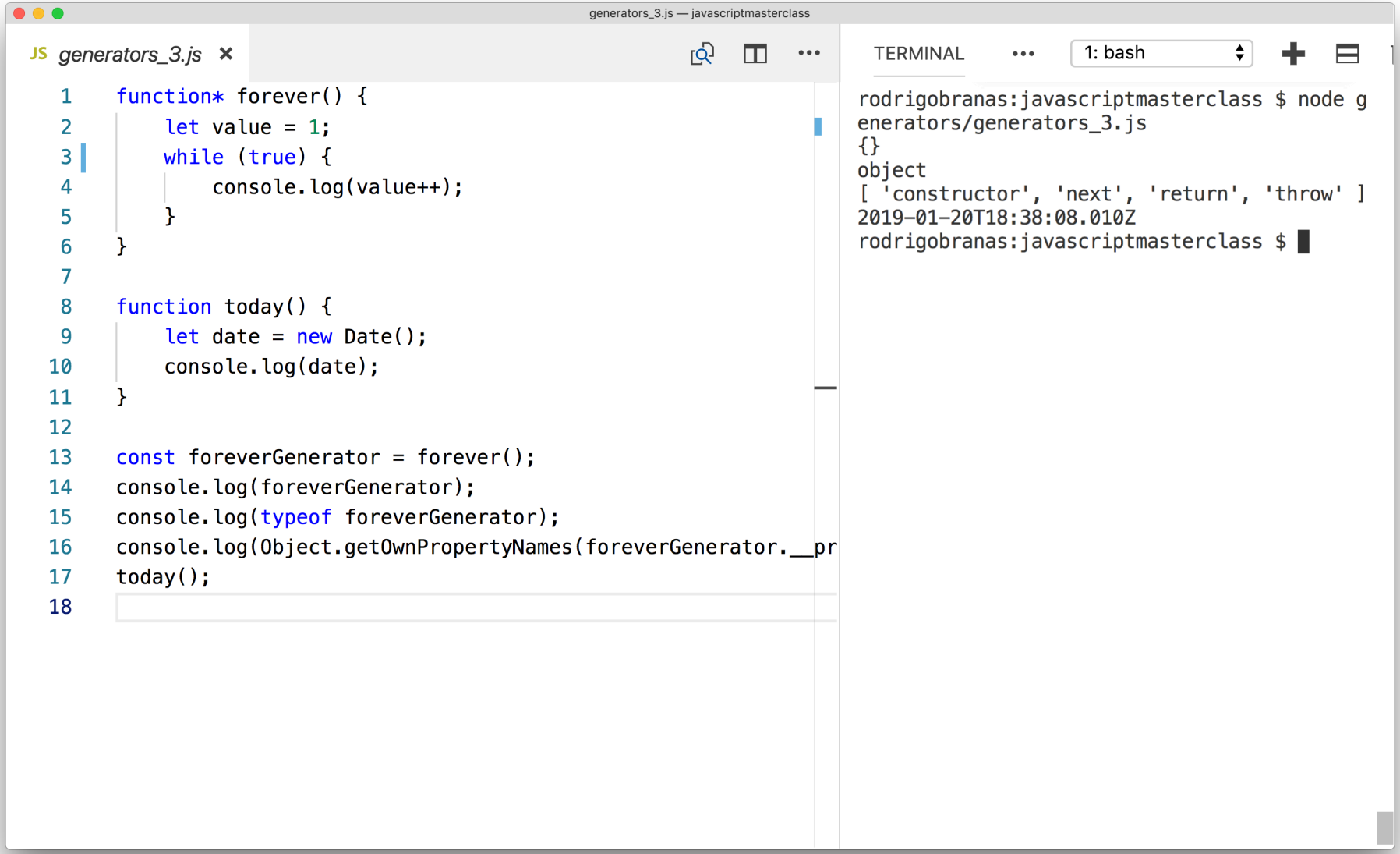

EJECT
REW/SEARCH
PLAY
FF/SEARCH
REC
PAUSE/STILL
STOP
SLOW
VHS
Video Cassette Recorder
SP LP SLP
SPEED

Generators

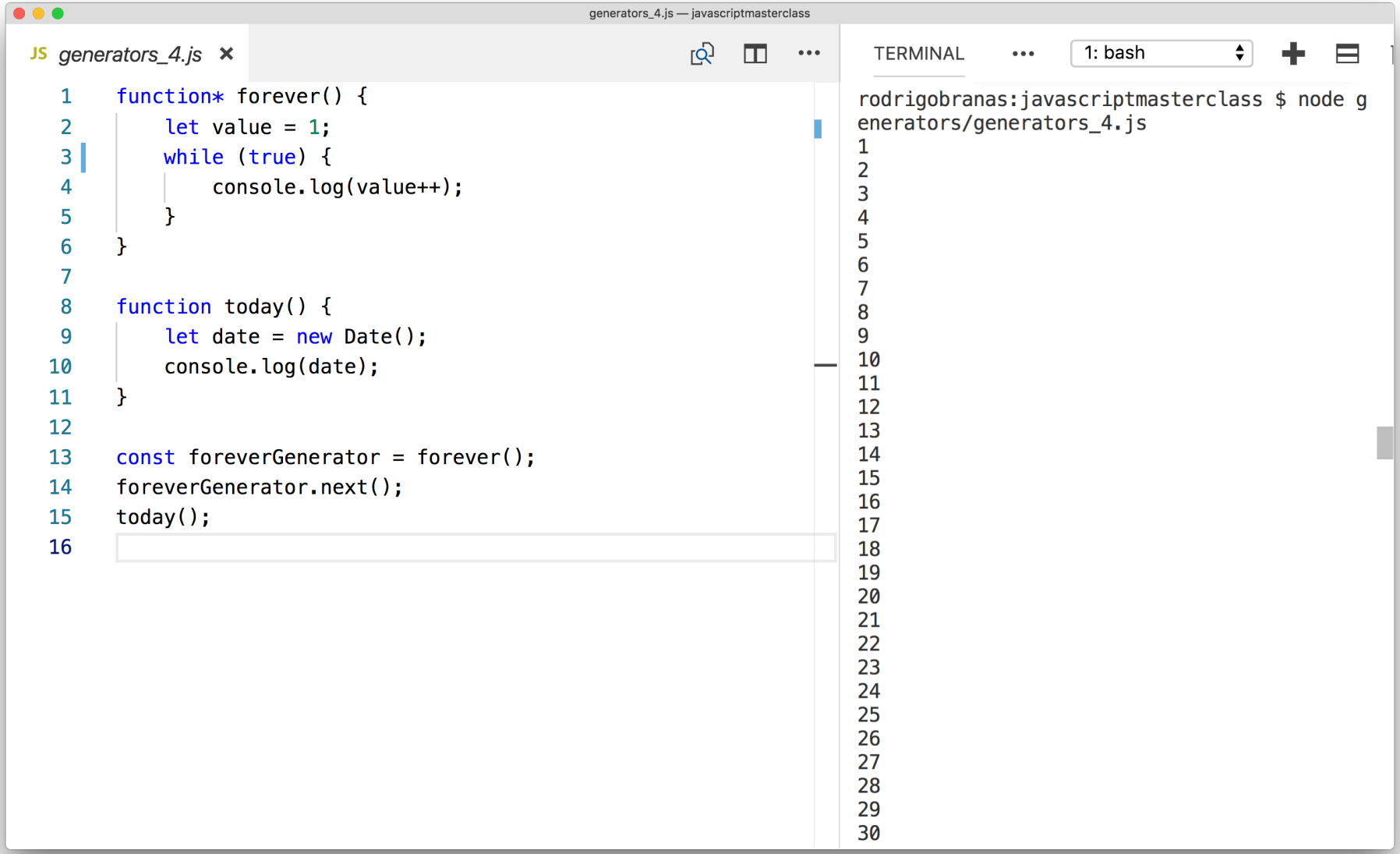
Os generators tornam possível pausar a execução de uma determinada função, permitindo a utilização do event loop de forma cooperativa


generators_1.js — javascriptmasterclass
generators_1.js
function forever() {
let value = 1;
while (true) {
console.log(value++);
}
}
function today() {
let date = new Date();
console.log(date);
}
forever();
today();
TERMINAL
1: bash
rodrigobranas:javascriptmasterclass $ node g
enerators/generators_1.js
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30

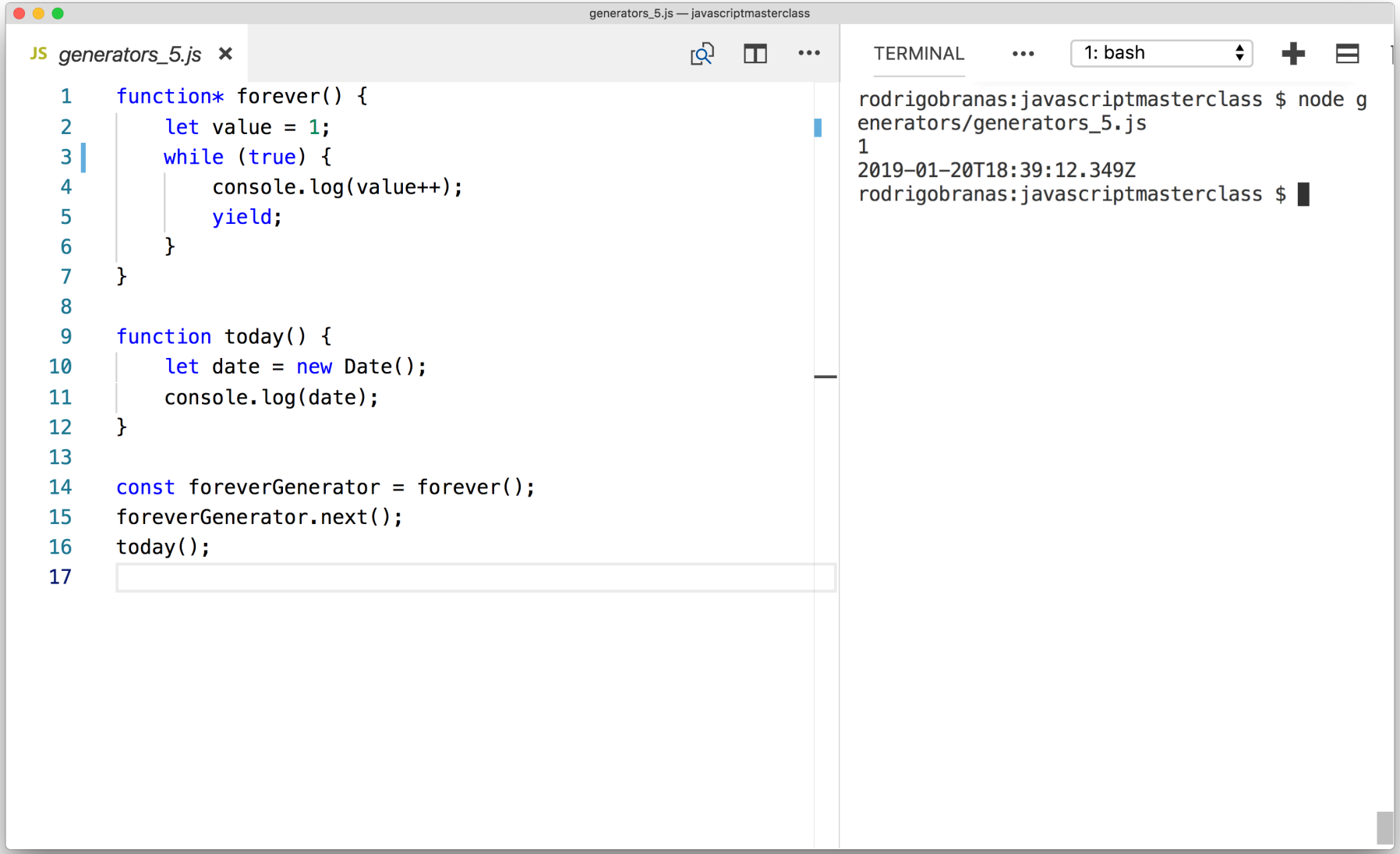

generators_2.js — javascriptmasterclass
generators_2.js
function* forever() {
let value = 1;
while (true) {
console.log(value++);
}
}
function today() {
let date = new Date();
console.log(date);
}
forever();
today();
TERMINAL
1: bash
rodrigobranas:javascriptmasterclass $ node g
enerators/generators_2.js
2019-01-20T18:37:58.476Z
rodrigobranas:javascriptmasterclass $

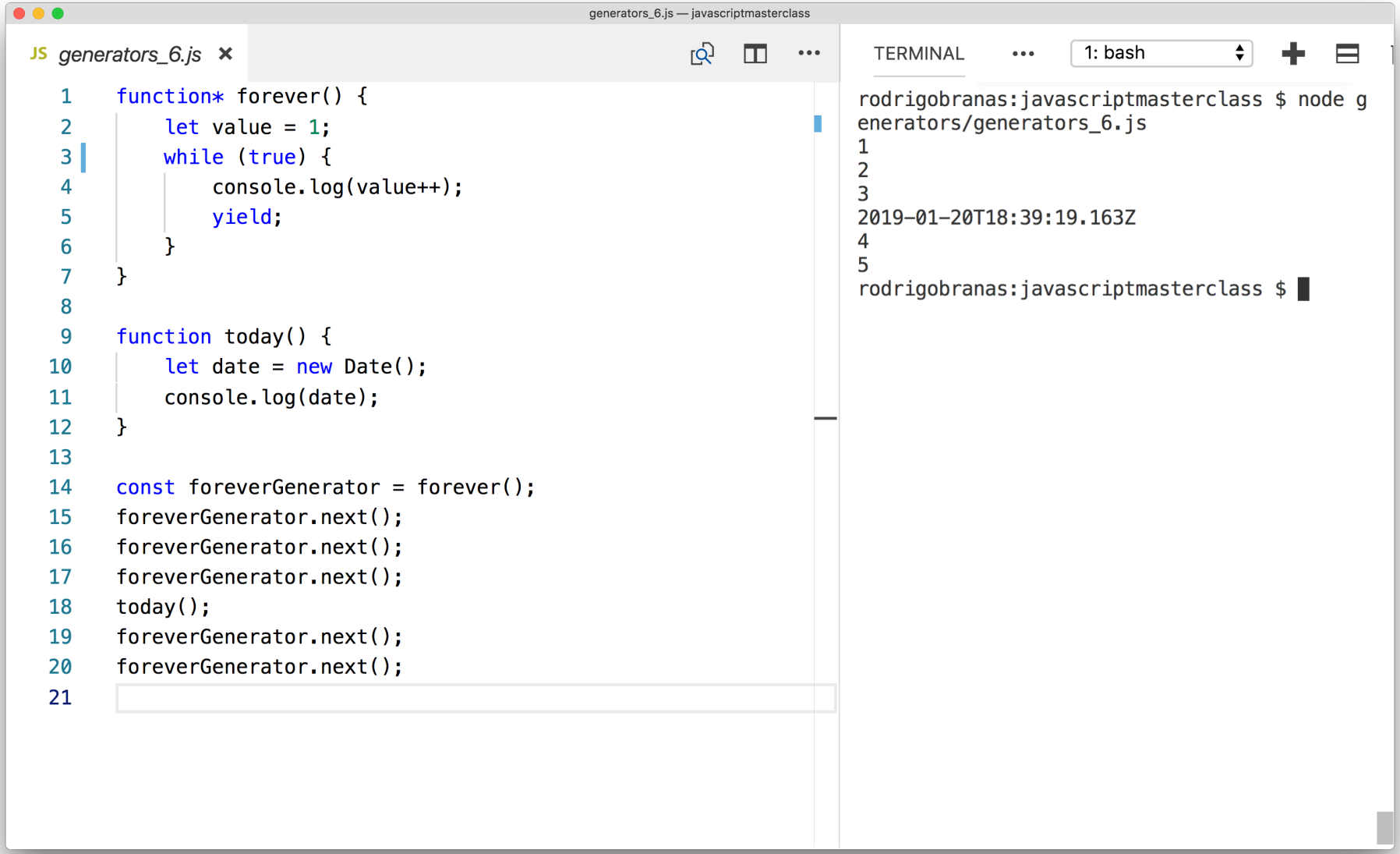

generators_3.js — javascriptmasterclass
generators_3.js
function* forever() {
let value = 1;
while (true) {
console.log(value++);
}
}
function today() {
let date = new Date();
console.log(date);
}
const foreverGenerator = forever();
console.log(foreverGenerator);
console.log(typeof foreverGenerator);
console.log(Object.getOwnPropertyNames(foreverGenerator.__pr
today();
TERMINAL
1: bash
rodrigobranas:javascriptmasterclass $ node g
enerators/generators_3.js
{}
object
['constructor', 'next', 'return', 'throw']
2019-01-20T18:38:08.010Z
rodrigobranas:javascriptmasterclass $

Os generators utilizam o método next para iterar sobre os valores disponíveis durante a execução da função

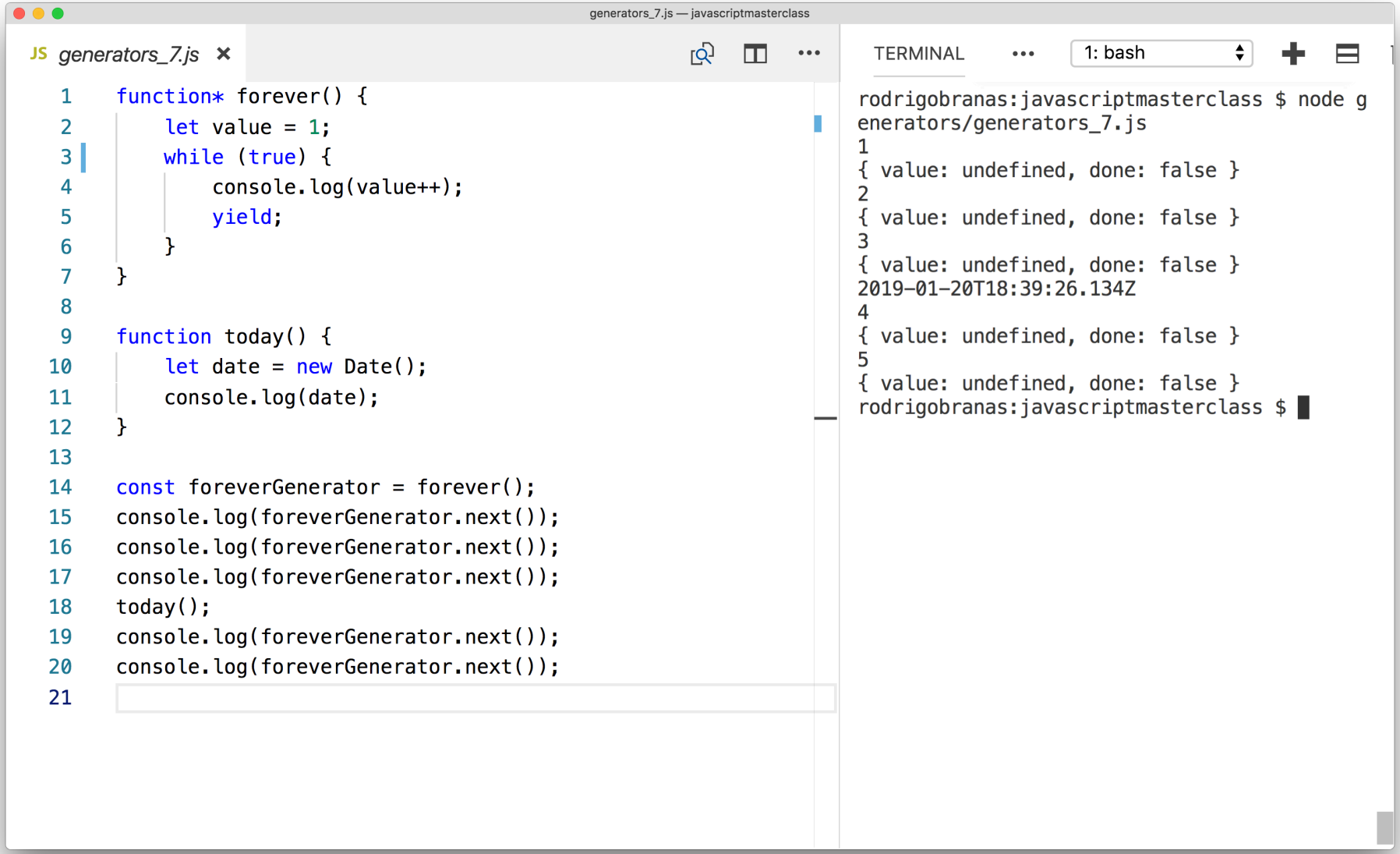

generators_4.js — javascriptmasterclass
generators_4.js
function* forever() {
let value = 1;
while (true) {
console.log(value++);
}
}
function today() {
let date = new Date();
console.log(date);
}
const foreverGenerator = forever();
foreverGenerator.next();
today();
TERMINAL
1: bash
rodrigobranas:javascriptmasterclass $ node g
enerators/generators_4.js
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30

Ao encontrar um yield, a execução da função é pausada até o método next ser invocado novamente

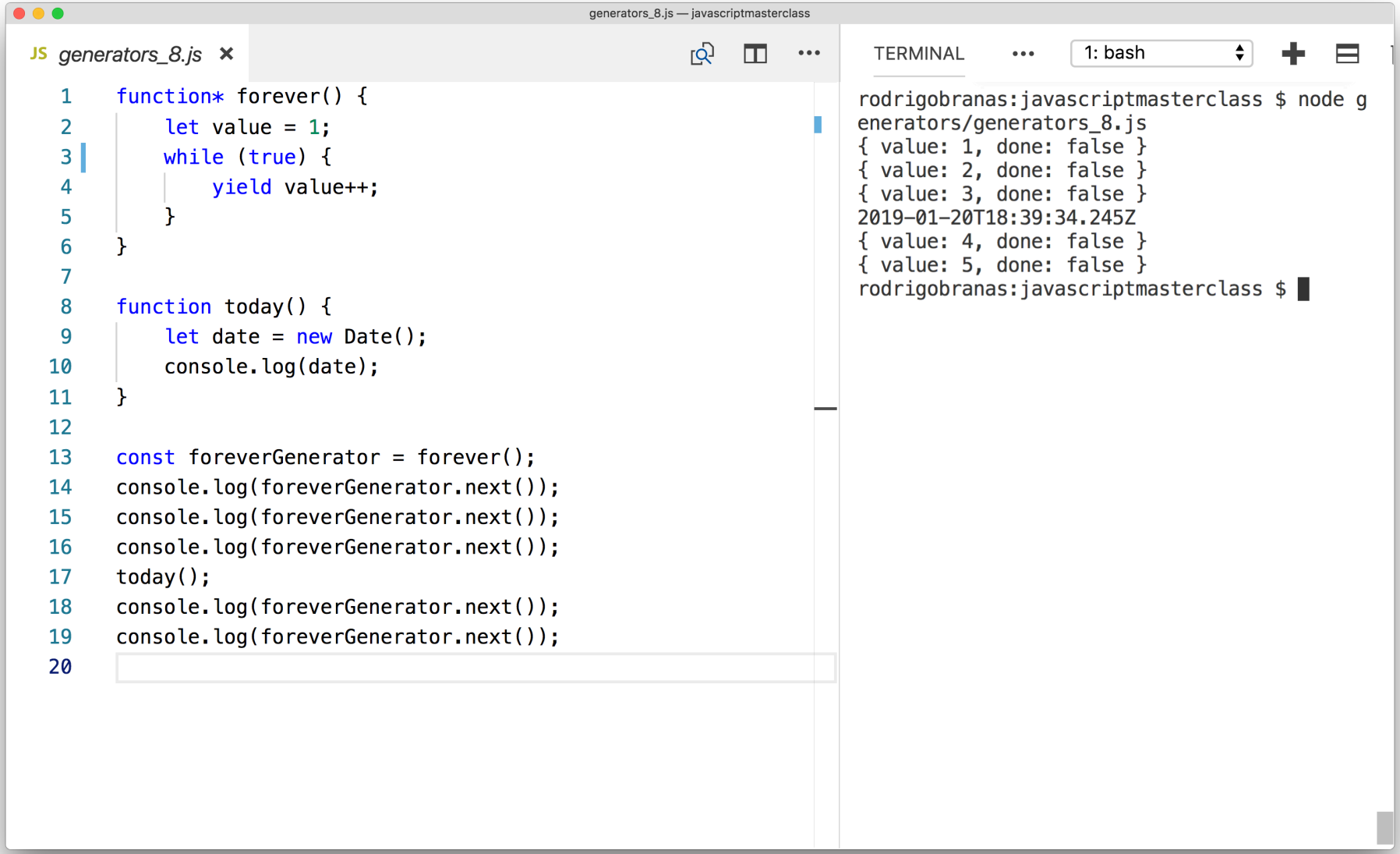

generators_5.js — javascriptmasterclass
generators_5.js
function* forever() {
let value = 1;
while (true) {
console.log(value++);
yield;
}
}
function today() {
let date = new Date();
console.log(date);
}
const foreverGenerator = forever();
foreverGenerator.next();
today();
TERMINAL
1: bash
rodrigobranas:javascriptmasterclass $ node g
enerators/generators_5.js
1
2019-01-20T18:39:12.349Z
rodrigobranas:javascriptmasterclass $


generators_6.js — javascriptmasterclass
generators_6.js
function* forever() {
let value = 1;
while (true) {
console.log(value++);
yield;
}
}
function today() {
let date = new Date();
console.log(date);
}
const foreverGenerator = forever();
foreverGenerator.next();
foreverGenerator.next();
foreverGenerator.next();
today();
foreverGenerator.next();
foreverGenerator.next();
TERMINAL
1: bash
rodrigobranas:javascriptmasterclass $ node g
enerators/generators_6.js
1
2
3
2019-01-20T18:39:19.163Z
4
5
rodrigobranas:javascriptmasterclass $

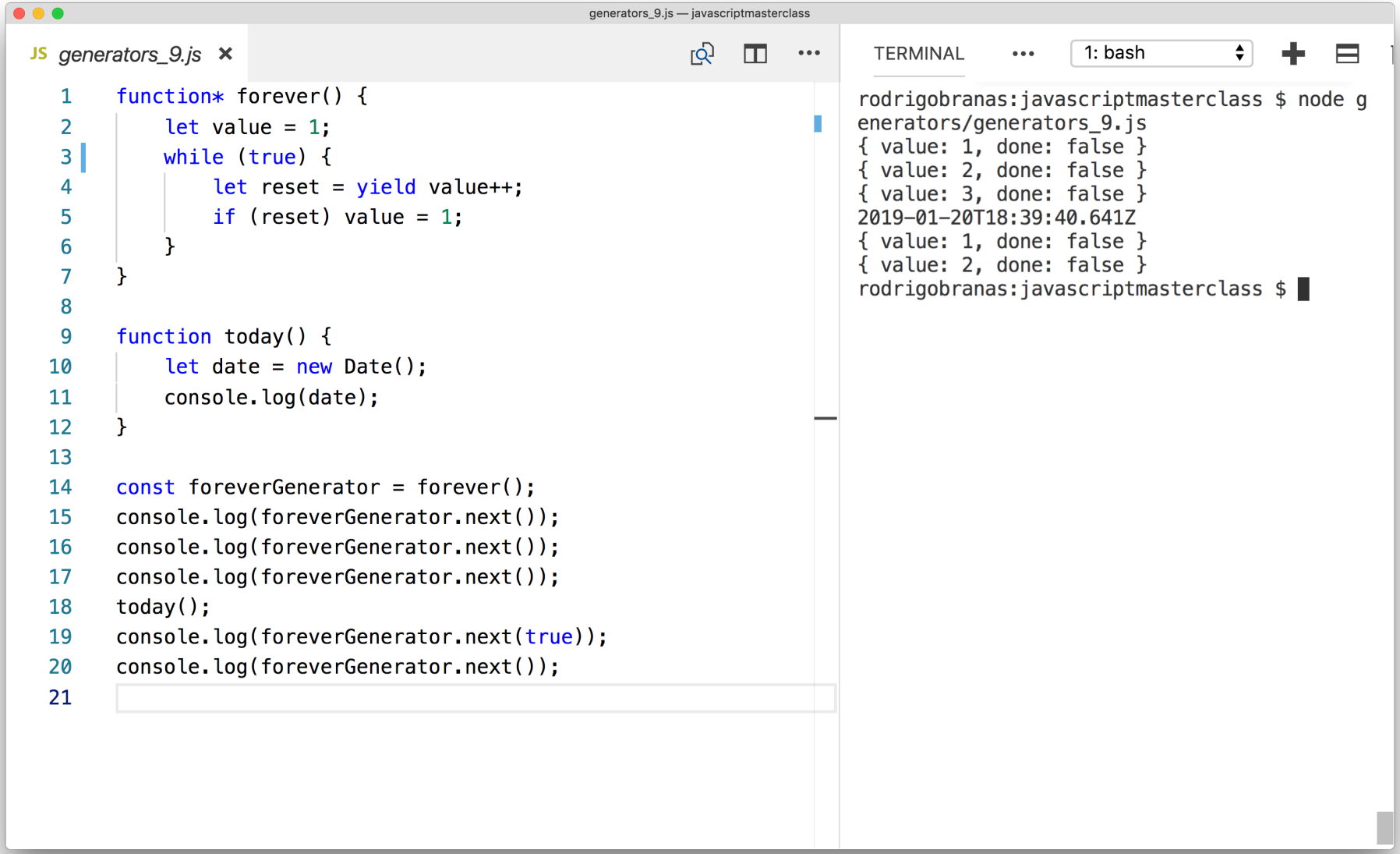
O retorno do método next é um objeto contendo value e done, seguindo o protocolo de iteração


generators_7.js — javascriptmasterclass
generators_7.js
function* forever() {
let value = 1;
while (true) {
console.log(value++);
yield;
}
}
function today() {
let date = new Date();
console.log(date);
}
const foreverGenerator = forever();
console.log(foreverGenerator.next());
console.log(foreverGenerator.next());
console.log(foreverGenerator.next());
today();
console.log(foreverGenerator.next());
console.log(foreverGenerator.next());
TERMINAL
1: bash
rodrigobranas:javascriptmasterclass $ node g
enerators/generators_7.js
1
{ value: undefined, done: false }
2
{ value: undefined, done: false }
3
{ value: undefined, done: false }
2019-01-20T18:39:26.134Z
4
{ value: undefined, done: false }
5
{ value: undefined, done: false }
rodrigobranas:javascriptmasterclass $

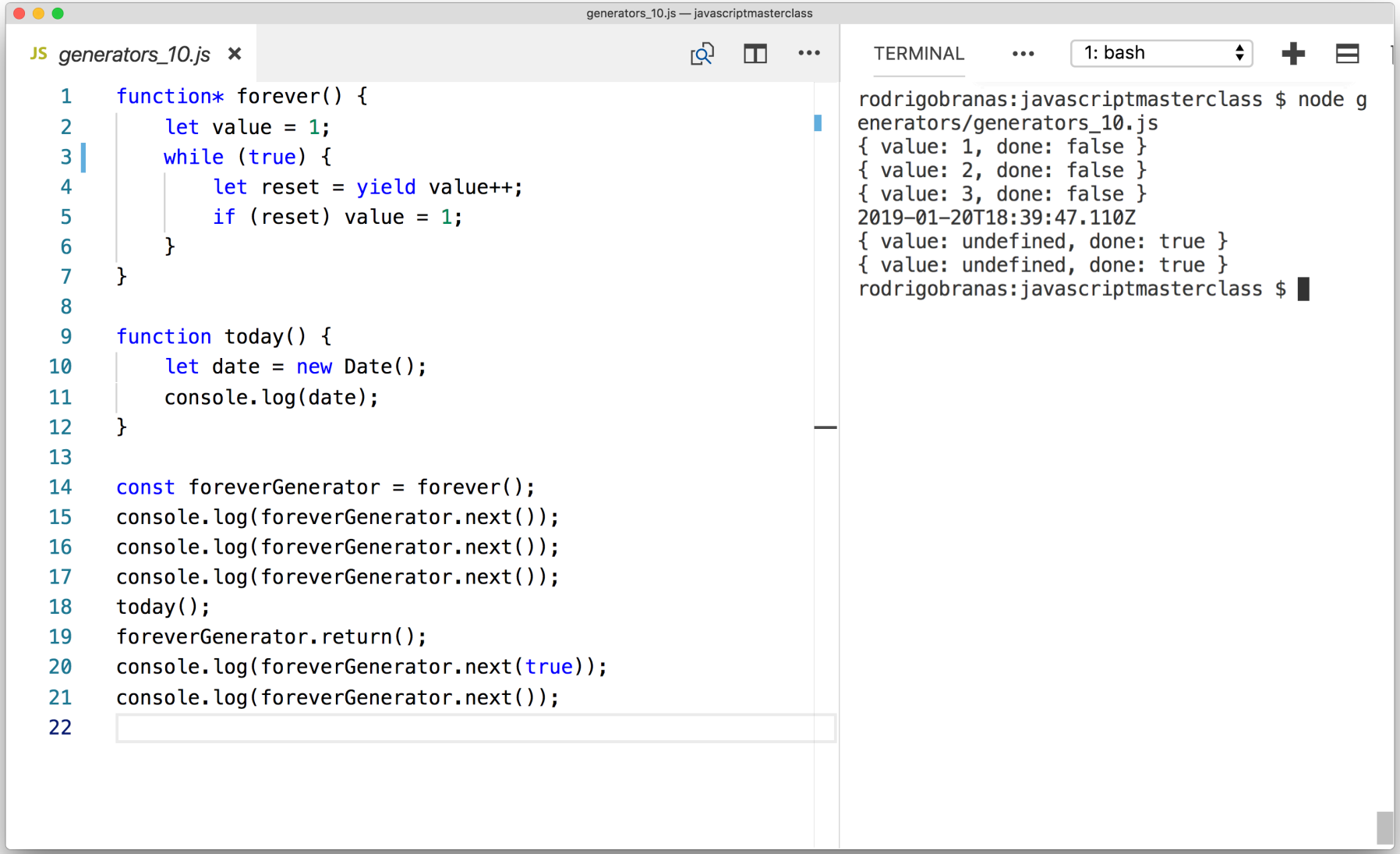
Por meio do yield é possível retornar valores de forma similar ao return

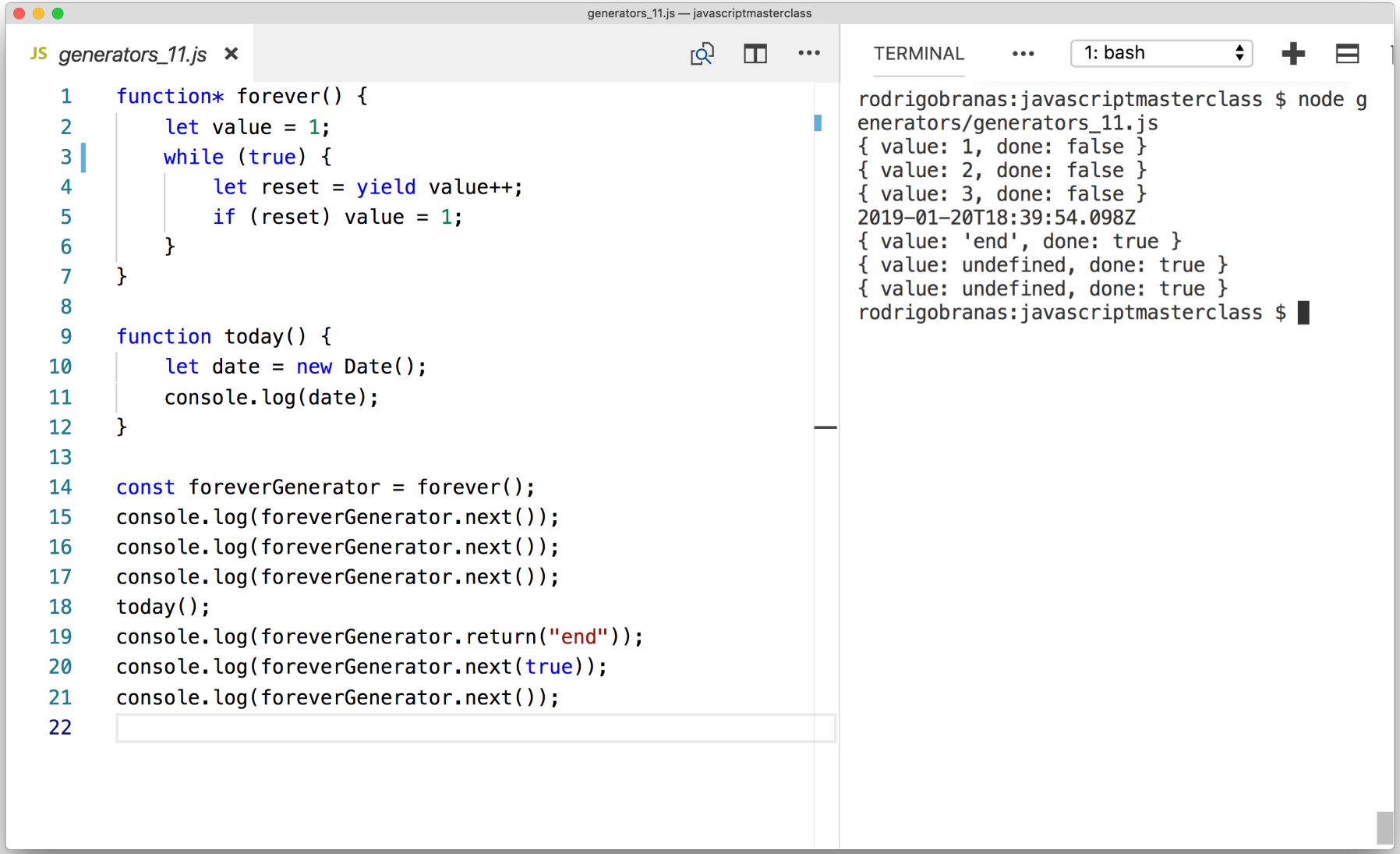

generators_8.js — javascriptmasterclass
generators_8.js
function* forever() {
let value = 1;
while (true) {
yield value++;
}
}
function today() {
let date = new Date();
console.log(date);
}
const foreverGenerator = forever();
console.log(foreverGenerator.next());
console.log(foreverGenerator.next());
console.log(foreverGenerator.next());
today();
console.log(foreverGenerator.next());
console.log(foreverGenerator.next());
TERMINAL
1: bash
rodrigobranas:javascriptmasterclass $ node g
enerators/generators_8.js
{ value: 1, done: false }
{ value: 2, done: false }
{ value: 3, done: false }
2019-01-20T18:39:34.245Z
{ value: 4, done: false }
{ value: 5, done: false }
rodrigobranas:javascriptmasterclass $

Além disso, também é possível enviar um valor para dentro do generator por meio do método next

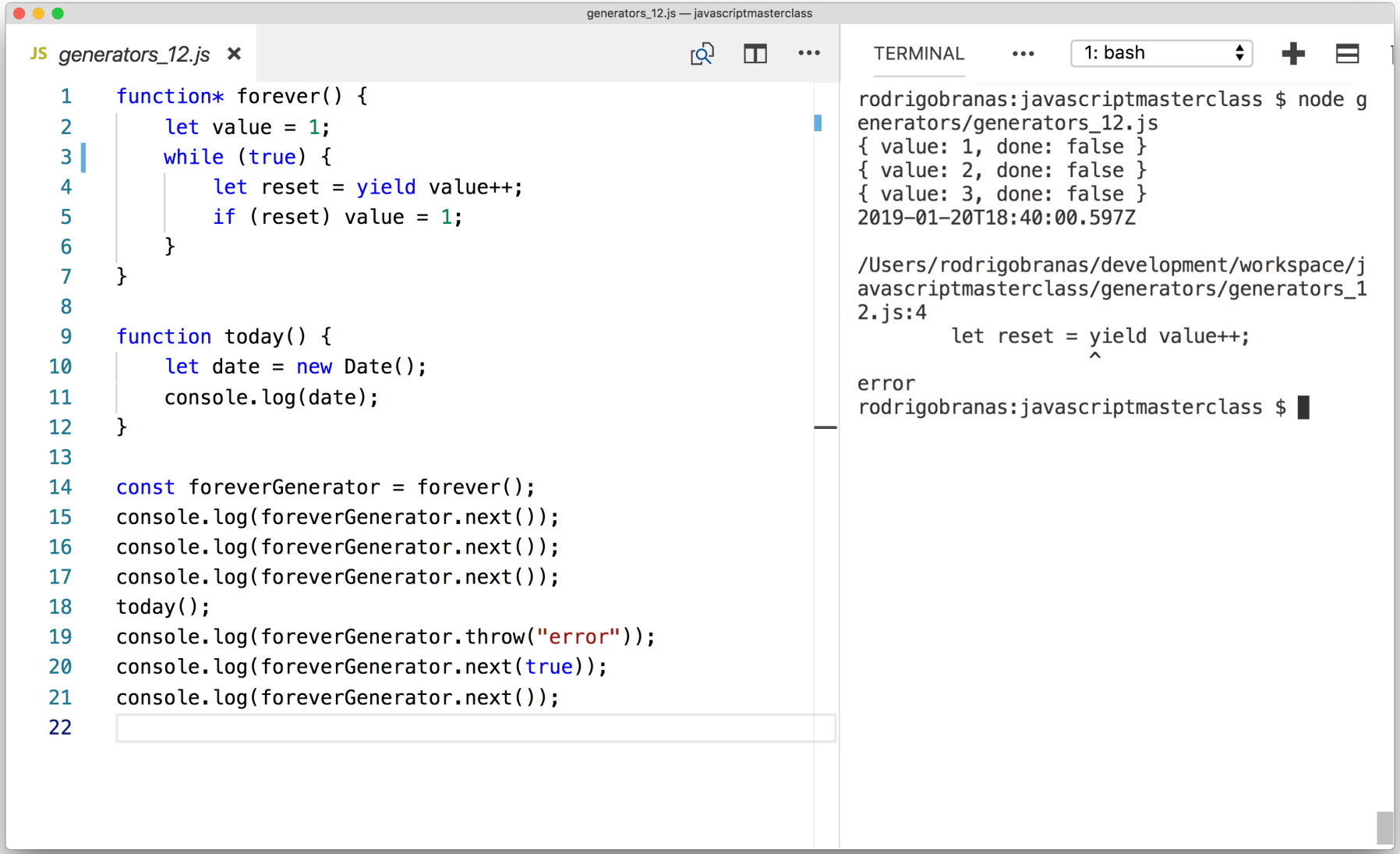

generators_9.js — javascriptmasterclass
generators_9.js
function* forever() {
let value = 1;
while (true) {
let reset = yield value++;
if (reset) value = 1;
}
}
function today() {
let date = new Date();
console.log(date);
}
const foreverGenerator = forever();
console.log(foreverGenerator.next());
console.log(foreverGenerator.next());
console.log(foreverGenerator.next());
today();
console.log(foreverGenerator.next(true));
console.log(foreverGenerator.next());
TERMINAL
1: bash
rodrigobranas:javascriptmasterclass $ node g
enerators/generators_9.js
{ value: 1, done: false }
{ value: 2, done: false }
{ value: 3, done: false }
2019-01-20T18:39:40.641Z
{ value: 1, done: false }
{ value: 2, done: false }
rodrigobranas:javascriptmasterclass $

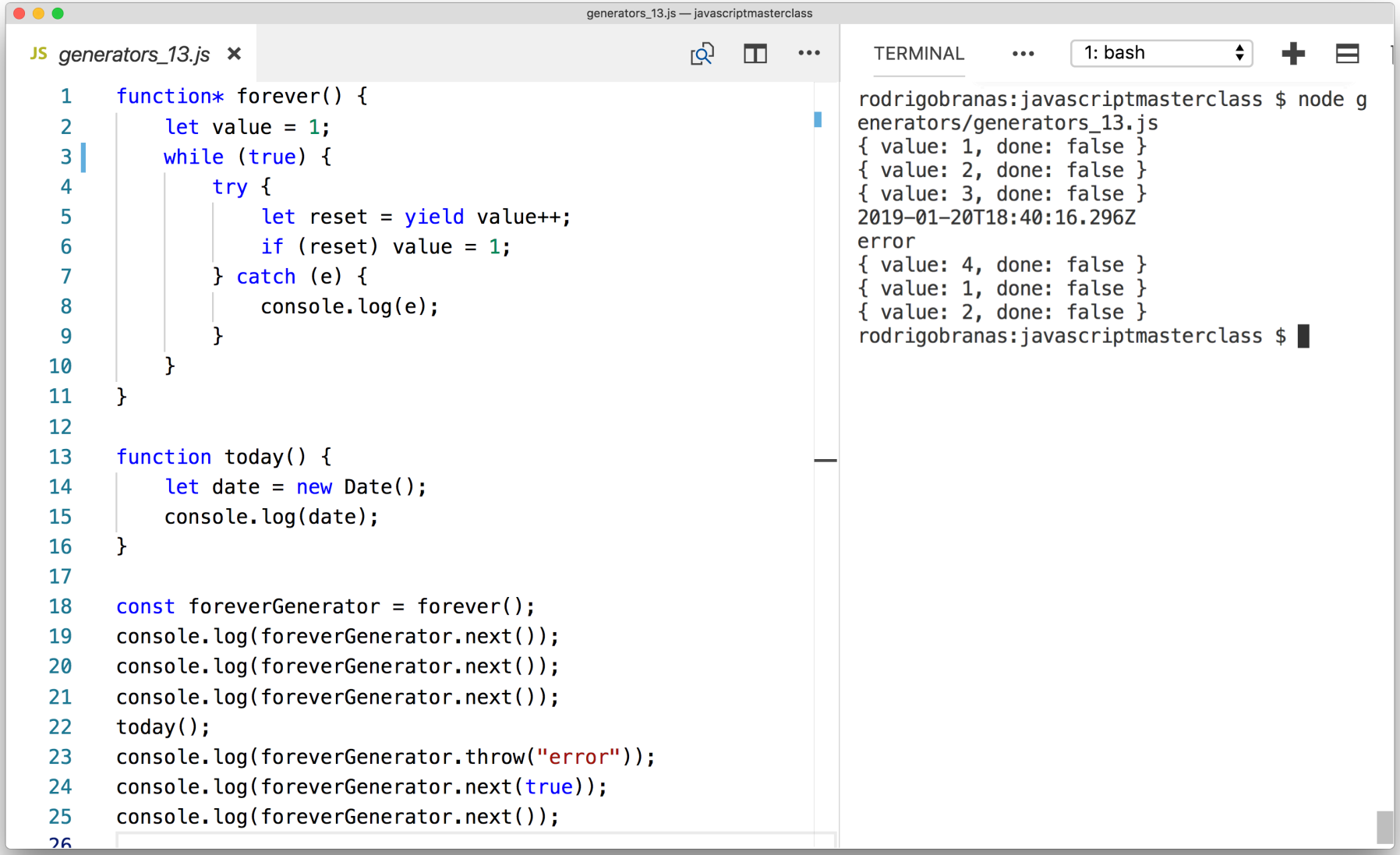
O método return encerra o generator podendo retornar um valor específico

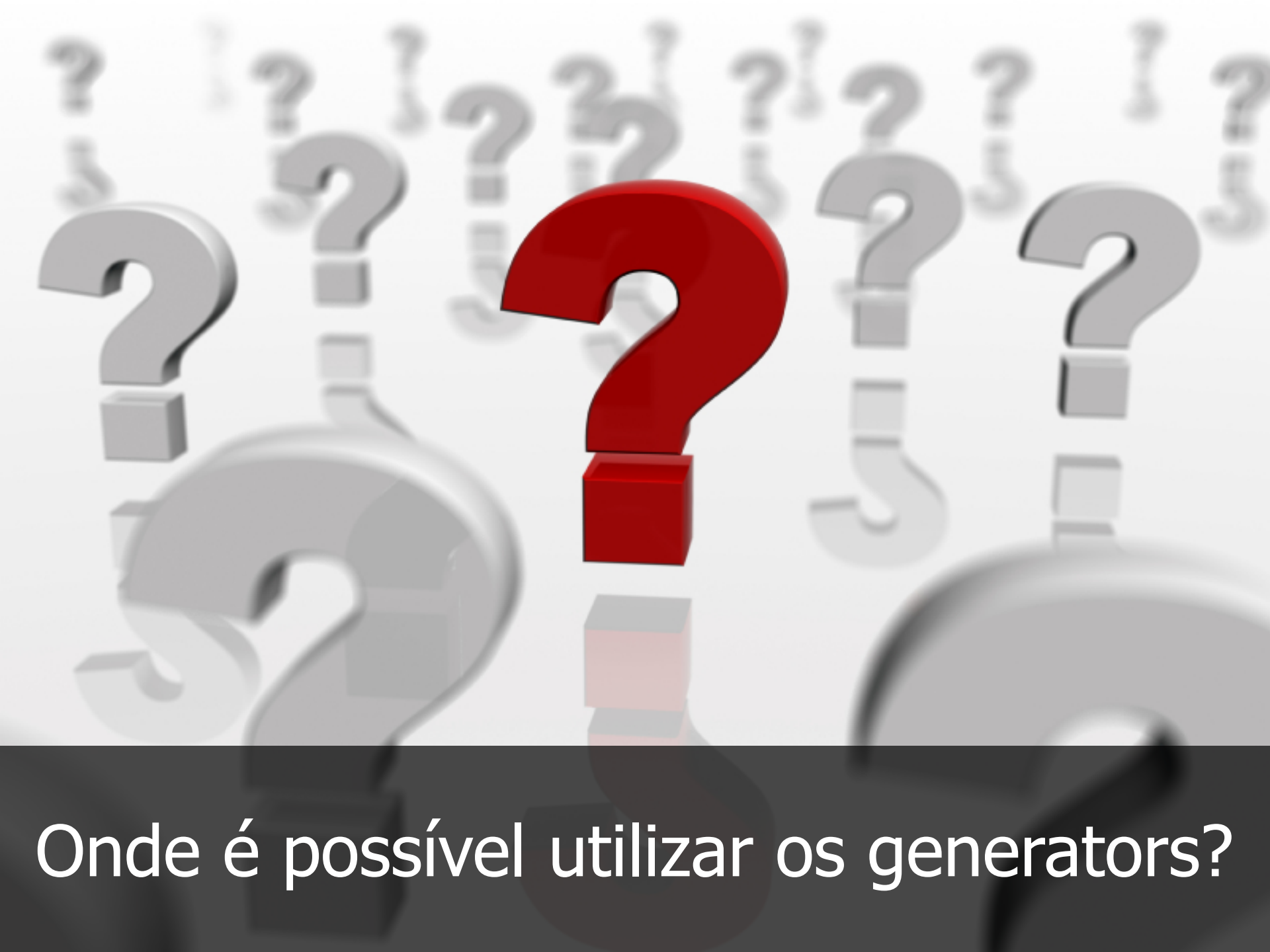

generators_10.js — javascriptmasterclass
generators_10.js
function* forever() {
let value = 1;
while (true) {
let reset = yield value++;
if (reset) value = 1;
}
}
function today() {
let date = new Date();
console.log(date);
}
const foreverGenerator = forever();
console.log(foreverGenerator.next());
console.log(foreverGenerator.next());
console.log(foreverGenerator.next());
today();
foreverGenerator.return();
console.log(foreverGenerator.next(true));
console.log(foreverGenerator.next());
TERMINAL
1: bash
rodrigobranas:javascriptmasterclass $ node g
enerators/generators_10.js
{ value: 1, done: false }
{ value: 2, done: false }
{ value: 3, done: false }
2019-01-20T18:39:47.110Z
{ value: undefined, done: true }
{ value: undefined, done: true }
rodrigobranas:javascriptmasterclass $


generators_11.js — javascriptmasterclass
generators_11.js
function* forever() {
let value = 1;
while (true) {
let reset = yield value++;
if (reset) value = 1;
}
}
function today() {
let date = new Date();
console.log(date);
}
const foreverGenerator = forever();
console.log(foreverGenerator.next());
console.log(foreverGenerator.next());
console.log(foreverGenerator.next());
today();
console.log(foreverGenerator.return("end"));
console.log(foreverGenerator.next(true));
console.log(foreverGenerator.next());
TERMINAL
1: bash
rodrigobranas:javascriptmasterclass $ node g
enerators/generators_11.js
{ value: 1, done: false }
{ value: 2, done: false }
{ value: 3, done: false }
2019-01-20T18:39:54.098Z
{ value: 'end', done: true }
{ value: undefined, done: true }
{ value: undefined, done: true }
rodrigobranas:javascriptmasterclass $

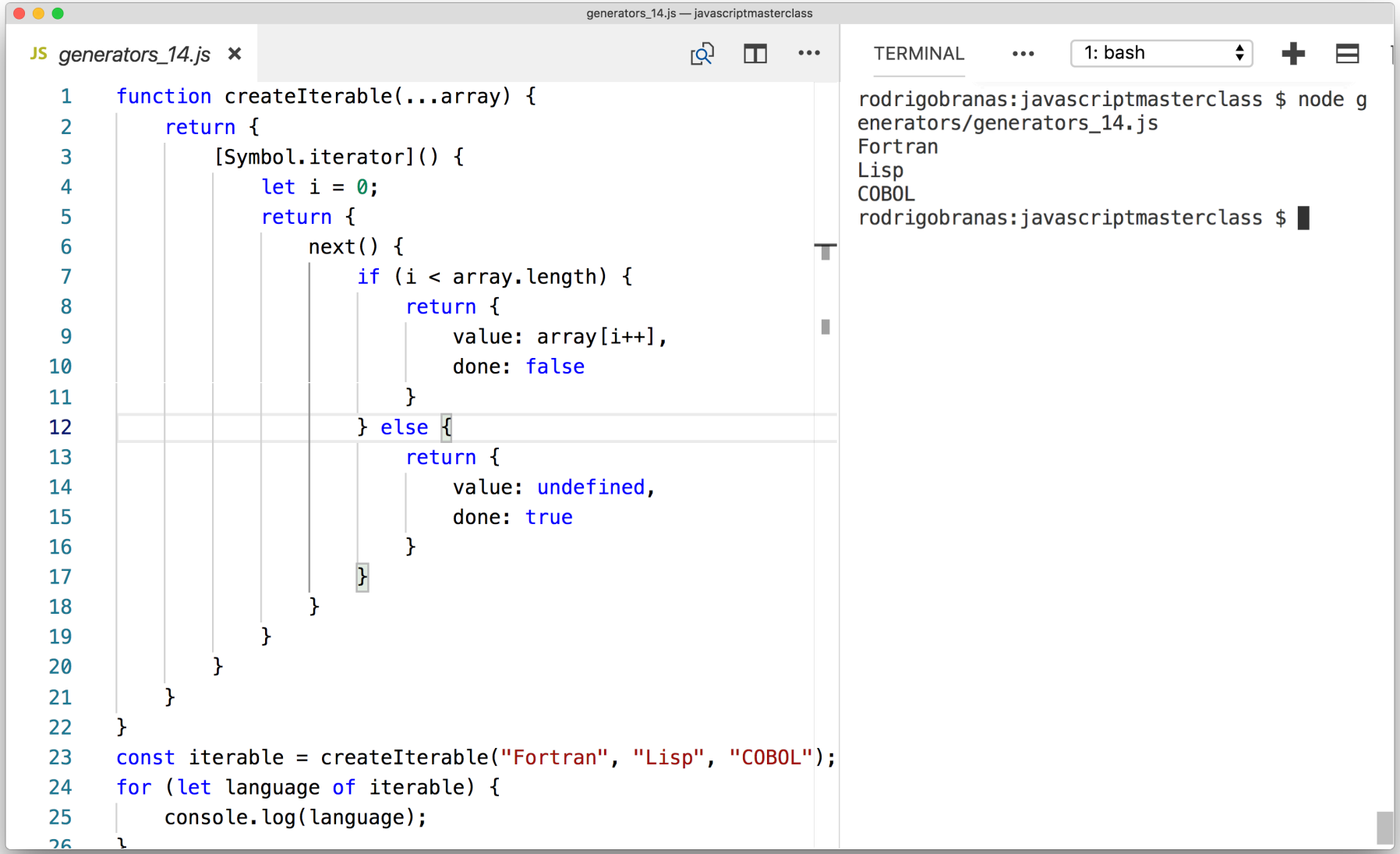
O método throw lança uma exceção dentro do generator interrompendo o fluxo de execução caso a exceção não tenha sido tratada adequadamente

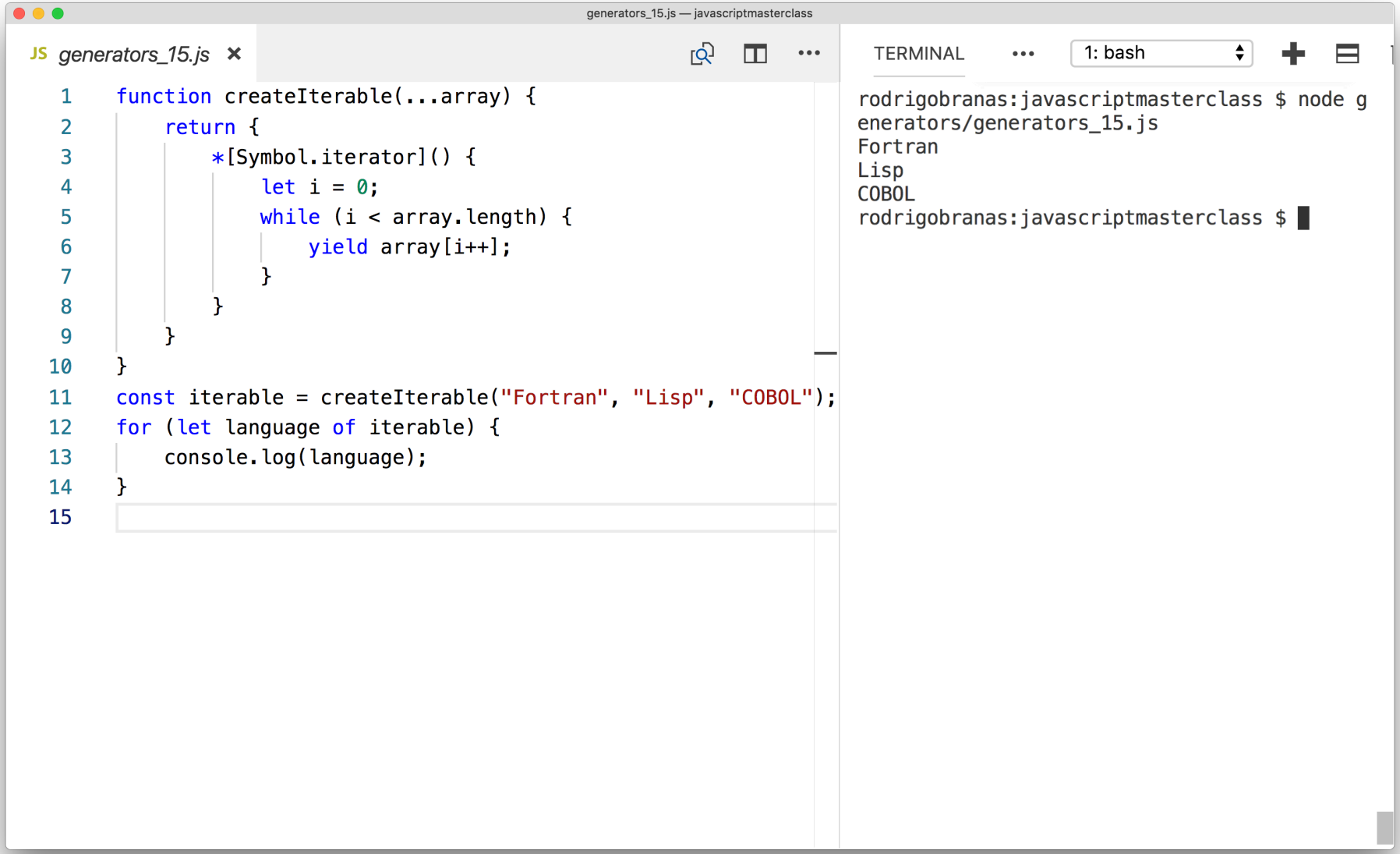

generators_12.js — javascriptmasterclass
generators_12.js
function* forever() {
let value = 1;
while (true) {
let reset = yield value++;
if (reset) value = 1;
}
}
function today() {
let date = new Date();
console.log(date);
}
const foreverGenerator = forever();
console.log(foreverGenerator.next());
console.log(foreverGenerator.next());
console.log(foreverGenerator.next());
today();
console.log(foreverGenerator.throw("error"));
console.log(foreverGenerator.next(true));
console.log(foreverGenerator.next());
TERMINAL
1: bash
rodrigobranas:javascriptmasterclass $ node g
enerators/generators_12.js
{ value: 1, done: false }
{ value: 2, done: false }
{ value: 3, done: false }
2019-01-20T18:40:00.597Z
/Users/rodrigobranas/development/workspace/j
avascriptmasterclass/generators/generators_1
2.js:4
let reset = yield value++;
^
error
rodrigobranas:javascriptmasterclass $


generators_13.js — javascriptmasterclass
generators_13.js
TERMINAL
1: bash
function* forever() {
let value = 1;
while (true) {
try {
let reset = yield value++;
if (reset) value = 1;
} catch (e) {
console.log(e);
}
}
}
function today() {
let date = new Date();
console.log(date);
}
const foreverGenerator = forever();
console.log(foreverGenerator.next());
console.log(foreverGenerator.next());
console.log(foreverGenerator.next());
today();
console.log(foreverGenerator.throw("error"));
console.log(foreverGenerator.next(true));
console.log(foreverGenerator.next());
rodrigobranas:javascriptmasterclass $ node g
enerators/generators_13.js
{ value: 1, done: false }
{ value: 2, done: false }
{ value: 3, done: false }
2019-01-20T18:40:16.296Z
error
{ value: 4, done: false }
{ value: 1, done: false }
{ value: 2, done: false }
rodrigobranas:javascriptmasterclass $

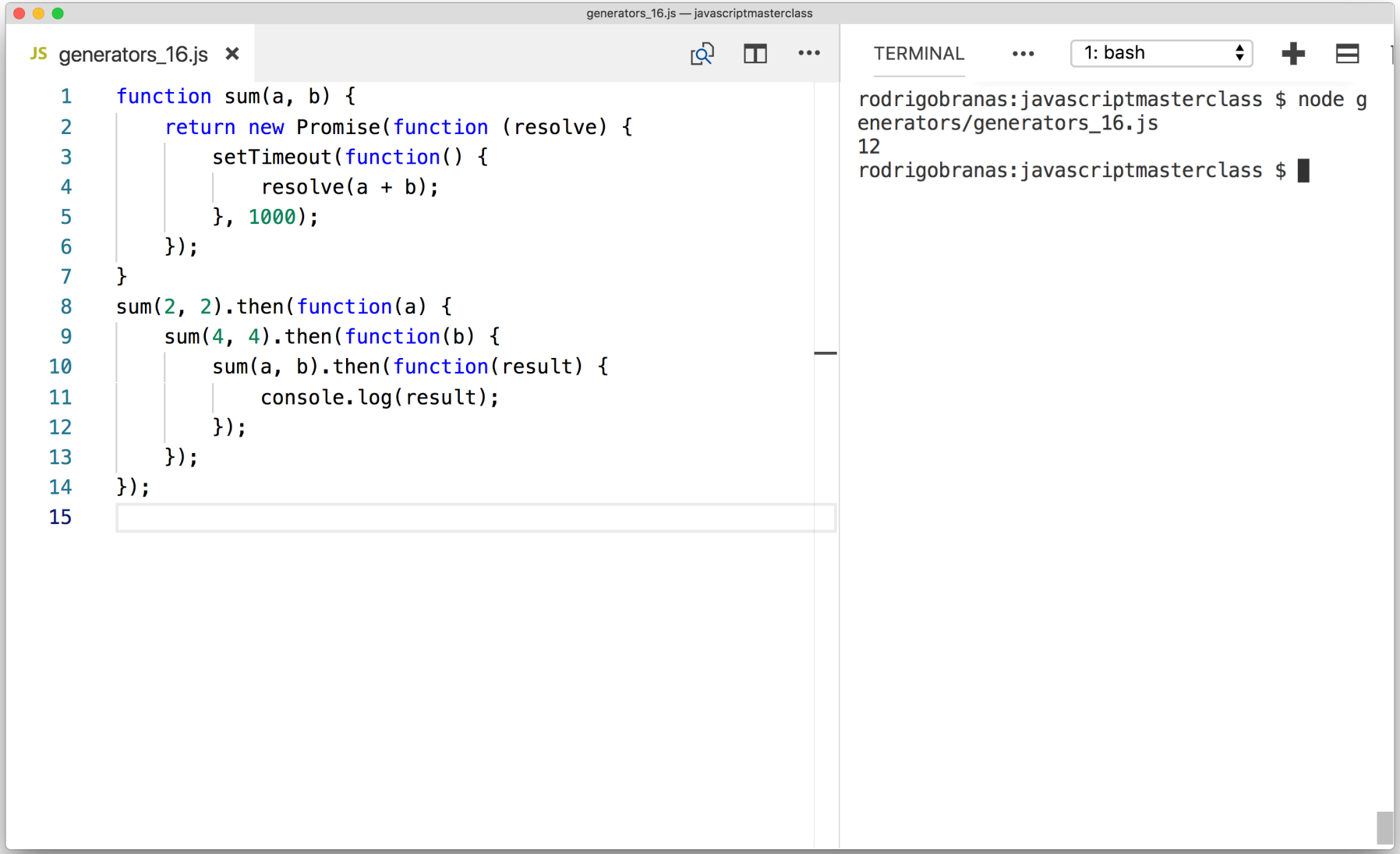

Onde é possível utilizar os generators?

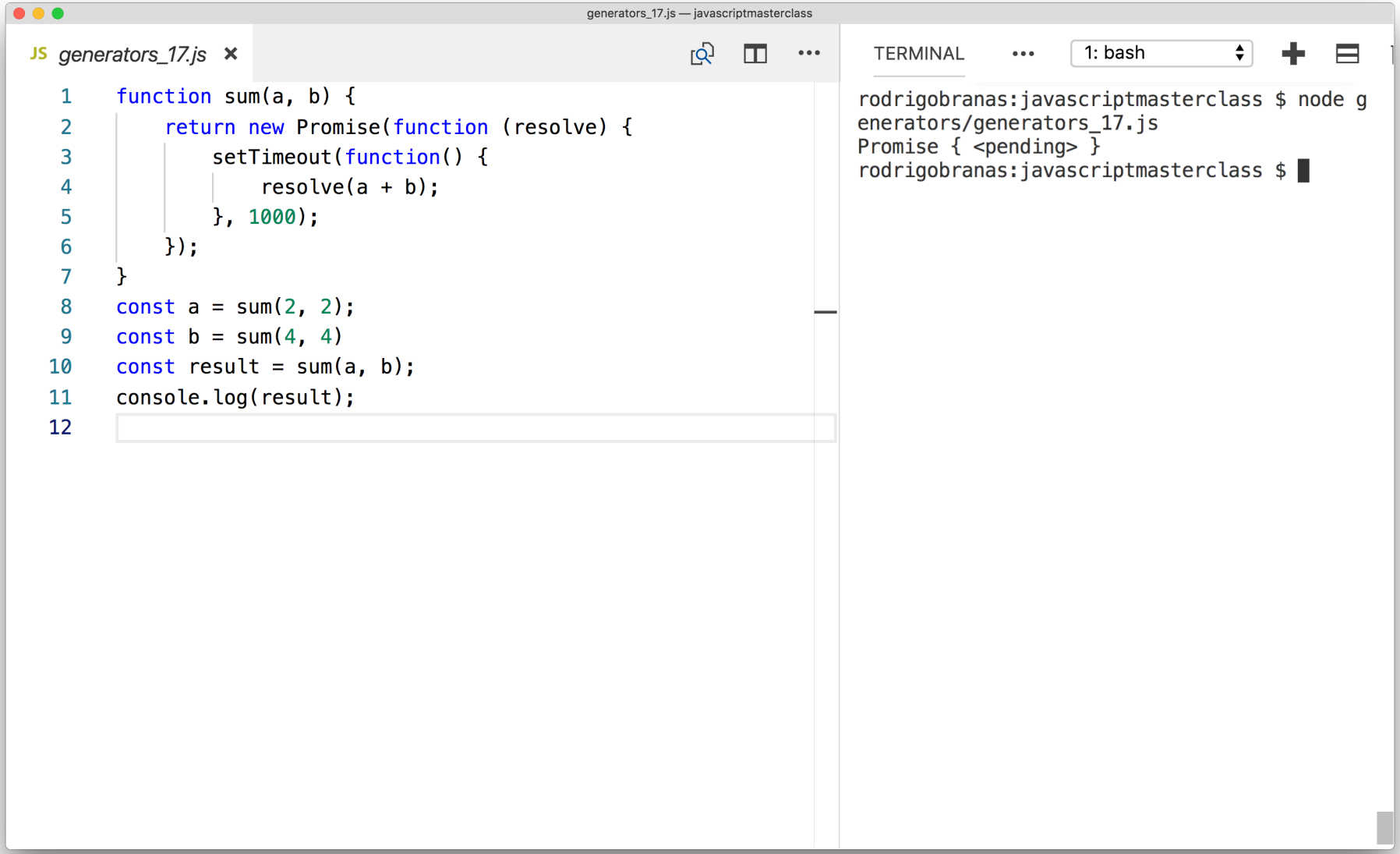
Como os generators implementam o protocolo de iteração é possível utilizá-los com Symbol.iterator de forma simples

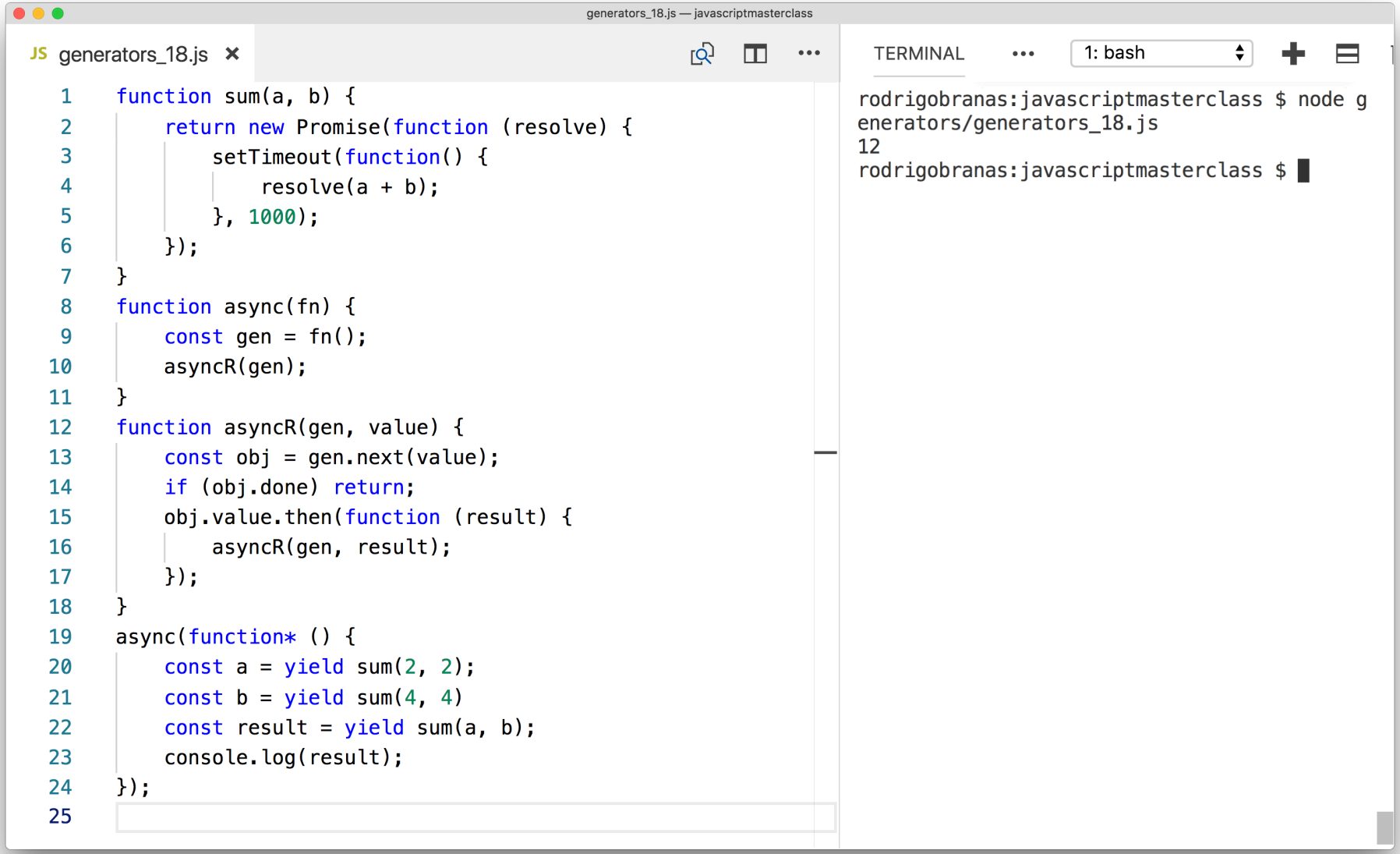

generators_14.js — javascriptmasterclass
generators_14.js
function createIterable(...array) {
return {
[Symbol.iterator]() {
let i = 0;
return {
next() {
if (i < array.length) {
return {
value: array[i++],
done: false
}
} else {
return {
value: undefined,
done: true
}
}
}
}
}
}
const iterable = createIterable("Fortran", "Lisp", "COBOL");
for (let language of iterable) {
console.log(language);
TERMINAL
1: bash
rodrigobranas:javascriptmasterclass $ node g
enerators/generators_14.js
Fortran
Lisp
COBOL
rodrigobranas:javascriptmasterclass $


generators_15.js — javascriptmasterclass
generators_15.js
function createIterable(...array) {
return {
*[Symbol.iterator]() {
let i = 0;
while (i < array.length) {
yield array[i++];
}
}
}
}
const iterable = createIterable("Fortran", "Lisp", "COBOL");
for (let language of iterable) {
console.log(language);
}
TERMINAL
1: bash
rodrigobranas:javascriptmasterclass $ node g
enerators/generators_15.js
Fortran
Lisp
COBOL
rodrigobranas:javascriptmasterclass $

Além disso, é possível utilizar generators para sincronizar chamadas assíncronas de forma similar ao async/await


generators_16.js — javascriptmasterclass
JS generators_16.js
function sum(a, b) {
return new Promise(function (resolve) {
setTimeout(function() {
resolve(a + b);
}, 1000);
});
}
sum(2, 2).then(function(a) {
sum(4, 4).then(function(b) {
sum(a, b).then(function(result) {
console.log(result);
});
});
});
TERMINAL
1: bash
rodrigobranas:javascriptmasterclass $ node g
enerators/generators_16.js
12
rodrigobranas:javascriptmasterclass $


generators_17.js — javascriptmasterclass
JS generators_17.js
function sum(a, b) {
return new Promise(function (resolve) {
setTimeout(function() {
resolve(a + b);
}, 1000);
});
}
const a = sum(2, 2);
const b = sum(4, 4)
const result = sum(a, b);
console.log(result);
TERMINAL
1: bash
rodrigobranas:javascriptmasterclass $ node g
enerators/generators_17.js
Promise { <pending> }
rodrigobranas:javascriptmasterclass $




```
1   function sum(a, b) {
2       return new Promise(function (resolve) {
3           setTimeout(function() {
4               resolve(a + b);
5           }, 1000);
6       });
7   }
8   function async(fn) {
9       const gen = fn();
10      asyncR(gen);
11  }
12  function asyncR(gen, value) {
13      const obj = gen.next(value);
14      if (obj.done) return;
15      obj.value.then(function (result) {
16          asyncR(gen, result);
17      });
18  }
19  async(function* () {
20      const a = yield sum(2, 2);
21      const b = yield sum(4, 4)
22      const result = yield sum(a, b);
23      console.log(result);
24  });
25
```