

Les fonctions génératrices en JS

```
1  function* miseEnJambe() {  
2    yield 'Hello les Pixous !'  
3  }
```

Comme itérables

Comme itérables

```
1  function* compterJusquà3() {
2    yield 1
3    yield 2
4    yield 3
5  }
6
7  console.log([ ...compterJusquà3()]) // [1, 2, 3]
8  console.log(Array.from(compterJusquà3())) // [1, 2, 3]
9
10 for (const value of compterJusquà3()) console.log(value)
11 // 1
12 // 2
13 // 3
14
15 const itérateur = compterJusquà3()
16 console.log(itérateur.next()) // { value: 1, done: false }
17 console.log(itérateur.next()) // { value: 2, done: false }
18 console.log(itérateur.next()) // { value: 3, done: false }
19 console.log(itérateur.next()) // { value: undefined, done: true }
```

Comme itérables

```
1  function* range(start, end, step = 1) {
2    for (let i = start; i ≤ end; i += step) yield i
3  }
4
5  console.log([ ... range(1, 9, 2)])           // [1, 3, 5, 7, 9]
6  console.log(Array.from(range(0, 4), x ⇒ 2 ** x)) // [1, 2, 4, 8, 16]
7
8  function* fibonacci(n) {
9    for (let [i, a, b] = [0, 0, 1]; i < n; [i, a, b] = [i + 1, b, a + b]) yield a
10 }
11
12 console.log([ ... fibonacci(10)]) // [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Syntaxes

Syntaxes : fonctions et méthodes

```
1  function* nommée() {  
2    // ...  
3  }  
4  
5  const anonyme = function*() {  
6    // ...  
7  }  
8  
9  class Exemple {  
10    * méthode() {  
11      // ...  
12    }  
13  }  
14  
15  const exemple = {  
16    * méthode() {  
17      // ...  
18    }  
19  }
```

Syntaxes : fonctions fléchées

```
1 // Irregular
2 () =>* ...
3
4 // not the same order as in regular generator functions
5 () =>* ...
6
7 // also wrong order
8 () *=> ...
9
10 // ASI hazard
11 *() => ...
```

🕒 Mot clé *generator*? (à l'étape 1 au tc39)

```
1 generator function() {}
2
3 const foo = generator () => {}
```

Syntaxes : yield

```
1  function* exemple() {
2    yield 123
3    yield 'foo'
4    yield { x: 1, y: 2, z: 3 }
5
6    yield
7    yield undefined
8
9    const valeur = yield 'chercher la valeur'
10 }
11
12 console.log([ ...exemple()])
13 // [123, 'foo', { x: 1, y: 2, z: 3 }, undefined, undefined, 'chercher la valeur']
```


Syntaxes : return

```
1  function* renvoyerLaRéponse() {  
2    return 42  
3  }  
4  
5  const itérateur = renvoyerLaRéponse()  
6  console.log(itérateur.next()) // { value: 42, done: true }  
7  
8  console.log([ ...renvoyerLaRéponse() ]) // []
```

Syntaxes : yield*

```
1  function* exemple() {  
2    yield* [1, 2, 3]  
3  
4    yield* range(4, 6)  
5  
6    const réponse = yield* renvoyerLaRéponse()  
7    yield réponse  
8  }  
9  
10 console.log([ ...exemple()])  
11 // [1, 2, 3, 4, 5, 6, 42]
```

Syntaxes : génératrices et asynchrones

```
1  async function* getNombres() {
2    const res = await fetch('https://example.com/nombres')
3    const nombres = await res.json()
4    for (const n of nombres) yield n
5  }
6
7  const itérateur = getNombres()
8  console.log(await itérateur.next()) // { value: 42, done: false }
9  console.log(await itérateur.next()) // { value: 1024, done: false }
10 console.log(await itérateur.next()) // { value: undefined, done: true }
11
12 for await (const n of getNombres()) console.log(n)
13 // 42
14 // 1024
15
16 const exemple = {
17   async* méthode() {
18     // ...
19   }
20 }
```

The end ?

Générateurs

Générateurs

```
1  interface Generator {  
2      next(value: any): IteratorResult  
3      throw(e: any): IteratorResult  
4      return(value: any): IteratorResult  
5  }  
6  
7  interface IteratorResult {  
8      done: boolean  
9      value: any  
10 }
```

Générateurs

```
1
2
3  function* exemple() {
4
5      // ...
6
7      const resultat = yield 'tâche 1'
8      console.log(resultat) // 'OK'
9
10     try {
11         yield 'tâche 2'
12     } catch (e) {
13         console.error(e) // 'KO'
14     }
15
16     while (true) yield 'infini'
17
18     yield 'inaccessible'
19 }
```

```
1  const gen = exemple()
2
3  gen.next() // { value: 'tâche 1', done: false }
4
5  // ...
6
7  gen.next('OK') // { value: 'tâche 2', done: false }
8
9  // ...
10
11 gen.throw('KO') // { value: 'infini', done: false }
12
13
14
15
16 gen.return('STOP') // { value: 'STOP', done: true }
17
18 gen.next() // { value: undefined, done: true }
19
```

Runners / Schedulers

Runners / Schedulers : pour faire du async/await

co

task.js

```
1  import co from 'co'
2
3  function* main() {
4      const res = yield fetch('https://api.punkapi.com/v2/beers/106')
5      const data = yield res.json()
6      console.log(data)
7  }
8
9  co(main).catch(e => console.error(e))
```

Runners / Schedulers : pour les effets de bord

redux-saga

```
1  function* fetchUser(action) {
2    try {
3      const user = yield call(Api.fetchUser, action.payload.userId)
4      yield put({type: "USER_FETCH_SUCCEEDED", user: user})
5    } catch (e) {
6      yield put({type: "USER_FETCH_FAILED", message: e.message})
7    }
8  }
9
10 function* rootSaga() {
11   yield takeEvery("USER_FETCH_REQUESTED", fetchUser)
12 }
```

Runners / Schedulers : pour les effets de bord

Effection

```
1  import { fetch, main, withTimeout } from 'effection'
2
3  main(function*() {
4    let dayOfTheWeek = yield withTimeout(fetchWeekDay('est'), 1000)
5    console.log(`It is ${dayOfTheWeek}, my friends!`)
6  })
7
8  export function *fetchWeekDay(timezone) {
9    let response = yield fetch(`http://worldclockapi.com/api/json/${timezone}/now`)
10   let time = yield response.json()
11   return time.dayOfTheWeek
12 }
```

Runners / Schedulers : pour les effets de bord

Cuillere

```
1  async function inscrireEtudiant({ etudiant, formation }) {
2    return crud.transactional(client => {
3      await creerDossierEtudiant(etudiant, client)
4      await creerDossierFinancier(etudiant, client)
5      await inscrireEtudiantFormation(etudiant, formation, client)
6      // ...
7    })
8  }
9
10 function* inscrireEtudiant({ etudiant, formation }) {
11   yield creerDossierEtudiant(etudiant)
12   yield creerDossierFinancier(etudiant)
13   yield inscrireEtudiantFormation(etudiant, formation)
14   // ...
15 }
```

Runners / Schedulers : pour les effets de bord



Cuillere

```
1  async function recupererFormationEtudiant(etudiant, dataSources) {  
2      return dataSources. formations.get(etudiant.idFormation)  
3  }  
4  
5  function* recupererFormationEtudiant(etudiant) {  
6      return yield crud. formations.get(etudiant.idFormation)  
7  }
```

Issue 10238: Async generator: caught error on last yield is wrongly rethrown

```
1  async function* gen() {
2    try {
3      yield 42
4    } catch(e) {
5      console.log('Error caught!')
6    }
7  }
8
9  (async () => {
10    const g = gen()
11    await g.next() // go to yield 42
12    try {
13      await g.throw(new Error()) // throw error from the yield
14    } catch (e) {
15      console.error('e has been rethrown !')
16    }
17  })()
```

Inconvénients

- Nouvelles syntaxes / pratiques
- Nécessité d'un framework
- Mauvais support du typage
- Contamination du code

Pour aller plus loin

Effets algébriques