Les fonctions génératrices en JS

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

Comme itérables

Comme itérables

```
function* compterJusquà3() {
  yield 1
  yield 2
  yield 3
}

console.log([...compterJusquà3()])        // [1, 2, 3]
console.log(Array.from(compterJusquà3())) // [1, 2, 3]

for (const value of compterJusquà3()) console.log(value)
// 1
// 2
// 3

const itérateur = compterJusquà3()
console.log(itérateur.next()) // { value: 1, done: false }
console.log(itérateur.next()) // { value: 2, done: false }
console.log(itérateur.next()) // { value: 3, done: false }
console.log(itérateur.next()) // { value: undefined, done: true }
```

Comme itérables

```
function* range(start, end, step = 1) {
  for (let i = start; i ⩽ end; i += step) yield i
}

console.log([ ...range(1, 9, 2)])                    // [1, 3, 5, 7, 9]
console.log(Array.from(range(0, 4), x ⇒ 2 ** x)) // [1, 2, 4, 8, 16]

function* fibonacci(n) {
  for (let [i, a, b] = [0, 0, 1]; i < n; [i, a, b] = [i + 1, b, a + b]) yield a
}

console.log([ ...fibonacci(10)]) // [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Syntaxes

Syntaxes : fonctions et méthodes

```
function* nommée() {
  // ...
}

const anonyme = function*() {
  // ...
}

class Exemple {
  * méthode() {
    // ...
  }
}

const exemple = {
  * méthode() {
    // ...
  }
}
```

⛔ Syntaxes : fonctions fléchées

```
// Irregular
() =*> ...

// not the same order as in regular generator functions
() ⇒* ...

// also wrong order
() *⇒ ...

// ASI hazard
*() ⇒ ...
```

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

```
generator function() {}

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

```
function* renvoyerLaRéponse() {
  return 42
}

const itérateur = renvoyerLaRéponse()
console.log(itérateur.next()) // { value: 42, done: true }

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

```
async function* getNombres() {
  const res = await fetch('https://example.com/nombres')
  const nombres = await res.json()
  for (const n of nombres) yield n
}

const itérateur = getNombres()
console.log(await itérateur.next()) // { value: 42, done: false }
console.log(await itérateur.next()) // { value: 1024, done: false }
console.log(await itérateur.next()) // { value: undefined, done: true }

for await (const n of getNombres()) console.log(n)
// 42
// 1024

const exemple = {
  async* méthode() {
    // ...
  }
}
```

The end ?

Générateurs

Générateurs

```
interface Generator {
  next(value: any): IteratorResult
  throw(e: any): IteratorResult
  return(value: any): IteratorResult
}

interface IteratorResult {
  done: boolean
  value: any
}
```

Générateurs

```

function* exemple() {

  // ...

  const resultat = yield 'tâche 1'
  console.log(resultat) // 'OK'

  try {
    yield 'tâche 2'
  } catch (e) {
    console.error(e) // 'KO'
  }

  while (true) yield 'infini'

  yield 'inaccessible'
}
```

```
const gen = exemple()

gen.next() // { value: 'tâche 1', done: false }

// ...

gen.next('OK') // { value: 'tâche 2', done: false }

// ...

gen.throw('KO') // { value: 'infini', done: false }




gen.return('STOP') // { value: 'STOP', done: true }

gen.next() // { value: undefined, done: true }

```

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

```
function* fetchUser(action) {
  try {
    const user = yield call(Api.fetchUser, action.payload.userId)
    yield put({type: "USER_FETCH_SUCCEEDED", user: user})
  } catch (e) {
    yield put({type: "USER_FETCH_FAILED", message: e.message})
  }
}

function* rootSaga() {
  yield takeEvery("USER_FETCH_REQUESTED", fetchUser)
}
```

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

```
async function inscrireEtudiant({ etudiant, formation }) {
  return crud.transactional(client => {
    await creerDossierEtudiant(etudiant, client)
    await creerDossierFinancier(etudiant, client)
    await inscrireEtudiantFormation(etudiant, formation, client)
    // ...
  })
}

function* inscrireEtudiant({ etudiant, formation }) {
  yield creerDossierEtudiant(etudiant)
  yield creerDossierFinancier(etudiant)
  yield inscrireEtudiantFormation(etudiant, formation)
  // ...
}
```

Runners / Schedulers : pour les effets de bord

🥄 Cuillere

```
async function recupererFormationEtudiant(etudiant, dataSources) {
  return dataSources.formations.get(etudiant.idFormation)
}

function* recupererFormationEtudiant(etudiant) {
  return yield crud.formations.get(etudiant.idFormation)
}
```

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

```
async function* gen() {
  try {
    yield 42
  } catch(e) {
    console.log('Error caught!')
  }
}

(async () => {
  const g = gen()
  await g.next() // go to yield 42
  try {
    await g.throw(new Error()) // throw error from the yield
  } catch (e) {
    console.error('e has been rethrown !')
  }
})()
```

Inconvénients

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

Pour aller plus loin

Effets algébriques