Effects

Les effets de bord c'est tout ce qui interfère avec la pureté des fonctions (c-a-d le fait que f(x) = y, toujours, pour une même valeur de x). Cela peut être la lecture ou l'écriture dans un state global (par exemple une variable globale mutable), la lecture ou l'écriture dans un fichier, les exceptions, appels réseaux, etc. bref toutes les fonctions dans la catégorie **faire des choses** plutôt que **calculer des choses**.

Tail recursion to loops

- add a `whiletrue` loop
- create a variable, mutable, for the acc
- create a variable, mutable, for the current x value
- return when reaching the right x value
- add a throw exception at the end of the function

Recursion to tail Recursion

Create a helper function with an accumulator that takes mutable values.

Futures

- `A zip B` takes time `max(time(A), time(B))` and represents a parallel composition of asynchronous operations with a join.
- `A flatMap B` takes time `time(A) + time(B)` and represents sequential composition of asynchronous operations.
- `A; B` discards A and returns B. This means wait ignores A and runs only as along as B, which can even lead to the program exiting before A completes (this is not good!). Importantly, this does not run A and B sequentially: it starts them sequentially.

Promises

Les promises sont des objets que l'on créé pour décider de renvoyer une future. En fait notre fonction va créer une promesse, renvoyer la future associée à cette promesse et ensuite appeler `success` ou `failure` sur la promesse pour indiquer que la future est prête.

C'est comme `resolve, reject` en JavaScript.

```
import scala.concurrent.{ Future, Promise }
import scala.concurrent.ExecutionContext.Implicits.global

val p = Promise[T]()
val f = p.future

val producer = Future {
  val r = produceSomething()
  p.success(r)
  continueDoingSomethingUnrelated()
}

val consumer = Future {
  startDoingSomething()
  f.foreach { r =>
    doSomethingWithResult()
  }
}
```

Méthodes Future utiles

- `future1.map(f)` : renvoie une future qui est complétée quand `future1` est prêt et que `f(resultat1)` est prêt.
- `future1.flatMap(f)` : renvoie une future qui est complétée quand `future1` est prêt et que `f(resultat1)` est prêt (donc la seconde future est prête).
- `future1.zip(future2)` : renvoie une future qui est complétée quand les deux futures sont prêtes.
- `future1.sequence` : renvoie une future qui est complétée quand tous les futures dans la liste sont prêts.