Série d'exercices #$7\frac{1}{2}$

IFT-2035

June 1, 2025

Ceci, puis cela

Soit le code ci-dessous qui définit une opération *cpcMaybe*. Cette opération fait une sorte de composition de fonction similaire à une sorte de séquencement.

```
data Maybe α = Nothing | Just α

baseMaybe :: α → Maybe α
baseMaybe x = Just x

cpcMaybe :: Maybe α → (α → Maybe β) → Maybe β
cpcMaybe x f = case x of
   Nothing → Nothing
   Just x' → f x'
```

Cette opération peut être utilisée par exemple si on veut composer non pas deux fonctions $f :: \alpha \to \beta$ et $g :: \beta \to \gamma$ mais deux fonctions $f :: \alpha \to Maybe\ \beta$ et $g :: \beta \to Maybe\ \gamma$.

Définir des fonctions similaires pour les types suivants:

```
-- Renvoie une valeur ou une erreur.
data Err α = Err String | Suc α

baseErr :: α → Err α
cpcErr :: Err α → (α → Err β) → Err β

-- Une valeur avec un compteur du coût pour la calculer.
data CountSteps a = CountSteps Int α

baseCS :: α → CountSteps α
cpcCS :: CountSteps a → (α → CountSteps β) → CountSteps β
```

Finalement généraliser ce "design pattern" en définissant une *classe de type* (ou plus précisément une classe de *constructeurs de types*) *CPC f* avec des instances *CPC Maybe*, *CPC Err*, et *CPC CountSteps*.