MODÉLISER UNE ERREUR POTENTIELLE

RESULT

```
type Error<E> = { kind: "Error", error: E }
type Ok<A> = { kind: "Ok", ok: A }
type Result<E, A> = Error<E> | Ok<A>
const isNat = (x: number): Result<string, number> =>
  x > 0
    ? { kind: "Ok", ok: x }
    : { kind: "Error", error: "Not a natural number" }

const foldNumber = (y: Result<string, number>, defaultValue: number): number => {
  switch (y.kind) {
    case "Ok": return y.ok;
    case "Error": return defaultValue;
  }
}
```

Les données qui peuvent être en erreurs sont des types SOMME

MODÉLISER UNE ERREUR POTENTIELLE

EITHER = RESULT

```ts
type Left<E> = { kind: "Left", left: E }
type Right<A> = { kind: "Right", right: A }
type Either<E, A> = Left<E> | Right<A>
const isNat = (x: number): Either<string, number> =>
  x > 0
    ? { kind: "Right", right: x }
    : { kind: "Left", left: "Not a natural number" }

const foldNumber = (y: Either<string, number>, defaultValue: number): number => {
  switch (y.kind) {
    case "Right": return y.right;
    case "Left": return defaultValue;
  }
}
```

Either est en réalité plus générique car représente n’importe quel arbre binaire

Par convention, il est souvent utilisé avec Right (correct) contenant la valeur ok, et Left (abandon) contenant la valeur error