MODÉLISER UNE ERREUR POTENTIELLE

AVEC FP-TS

MIAGE M2 - QUALITÉ DU SI - THOMAS HAESSLÉ & QUENTIN BURG

FP-TS fournit des fonctions utilitaires telles que le match expressif

```ts
import * as E from 'fp-ts/Either'
import { Either } from 'fp-ts/Either'
import { pipe } from "fp-ts/lib/function";

const isNat = (x: number): Either<string, number> =>
  x > 0
    ? E.right(x)
    : E.left("Not a natural number")

const foldNumber = (y: Either<string, number>, defaultValue: number): number =>
  pipe(
    y,
    E.match(
      (_: string) => defaultValue, // onLeft handler
      (value: number) => value // onRight handler
    )
  )
```

MODÉLISER UNE ERREUR POTENTIELLE

AVEC FP-TS

```ts
import * as E from 'fp-ts/Either'
import { Either } from 'fp-ts/Either'
import { pipe } from "fp-ts/lib/function";

const isNat = (x: number): Either<string, number> =>
  x > 0
    ? E.right(x)
    : E.left("Not a natural number")

const foldNumber = (y: Either<string, number>, defaultValue: number): number =>
  pipe(
    y,
    E.match(
      (_: string) => defaultValue, // onLeft handler
      (value: number) => value // onRight handler
    )
  )
```

FP-TS fournit des fonctions utilitaires telles que le match expressif

MODÉLISER UNE ERREUR POTENTIELLE

CHAIN (AKA FLATMAP AKA BIND) / MAP

```
type weapon = string
type target = string
type impacted = { impacted: target }

let must_be_carried = (w : target) : Either<string,target> =>
    w === "bow" ? E.right(w) : E.left("not carried »)

let hit_monster = (w: Either<string, weapon>, t: Either<string, target>): Either<string, impacted> =>
  pipe(
    w,
    E.chain(
      (_carried) => pipe(
        t,
        E.map(
          (targeted) => ({ impacted: targeted })
        )
      )
    )
  )
```