# Scala vs. OCaml

```
sealed abstract class Expr
case class Var(s: String) extends Expr
case class Const(i: Int) extends Expr
case class Add(e1: Expr, e2: Expr) extends Expr
case class Mul(e1: Expr, e2: Expr) extends Expr
object Task {
  def derive(x: String, e: Expr): Expr = e match {
    case Const(_) => Const(0)
    case Var(y) => Const(if (x==y) 1 else 0)
    case Add(a, b) => Add(derive(x,a), derive(x,b))
    case Mul(a, b) => Add(Mul(derive(x,a), b), Mul(a, derive(x,b)))
}}
```

```
type expr =
| Var of string
| Const of int
| Add of expr * expr
| Mul of expr * expr

let rec derive x e = match e with
  | Const _ -> Const 0
  | Var y -> Const(if x = y then 1 else 0)
  | Add(a, b) -> Add(derive x a, derive x b)
  | Mul(a, b) -> Add(Mul(derive x a, b), Mul(a, derive x b))
```

# Scala vs. OCaml

```scala
sealed abstract class Calc
case class Const(i: Int) extends Calc
case class Div(c1: Calc, c2: Calc) extends Calc
object Task {
  def div(p: Int, q: Int): Option[Int] =
    if (q==0) None else Some(p / q)

  def eval(e: Calc): Option[Int] = e match {
    case Const(n) => Some(n)
    case Div(a,b) =>    eval(a).flatMap(va =>
                        eval(b).flatMap(vb =>
                        div(va,vb)))
}}
```

```ocaml
type calc =
| Const of int
| Div of calc * calc

let div p q : int option =
  if q = 0 then None else Some (p / q)

let rec eval e = match e with
| Const n -> Some n
| Div(a, b) ->          bind (eval a) (fun va ->
                        bind (eval b) (fun vb ->
                        div va vb))
```

# Scala vs. Java

**Langage fonctionnel (+ impératif)**

- Lambda expressions
- Récursivité
- Stream API
- Immutabilité
- Récursivité terminale optimisée
- Pattern-matching
- Curryfication
- « return » est implicite

**Supporte le style fonctionnel**

- Lambda expressions
- Récursivité
- Stream API (objets / types primitifs)
- final + bibliothèques 3rd-party
- Optimisation manuelle
- If-else
- Classe BiFunction
- « return » est explicite

# Scala vs. Java

**Langage fonctionnel (+ impératif)**

- Lambda expressions
- Récursivité
- Stream API
- Immutabilité
- Récursivité terminale optimisée

**Supporte le style fonctionnel**

- Lambda expressions
- Récursivité
- Stream API (objets / types primitifs)
- final + bibliothèques 3rd-party
- Optimisation manuelle

Les exercices montrent qu'on atteint les mêmes performances avec les deux langages, mais que cela demande plus d'efforts avec **Java**.

# Scala vs. Java

**Autres caractéristiques**

- Orienté Objet (tout est objet !)
- Typage statique
- Concis
  - inférence de type
  - implicites
  - ce qui ne sert pas à désambiguïser peut généralement être omis !
- Compatibilité non garantie
- Extensible (« Scalable »)

**Autres caractéristiques**

- Orienté Objet (sauf types primitifs)
- Typage statique
- Verbeux
  - inférence de type à partir de java 11
- Compilation plus rapide que Scala
- Compatibilité ascendante garantie
- Très utilisé (sauf chez nous…)

# Scala + Java

- Interopérabilité dans les 2 sens (Java → Scala et Scala → Java)
  - Chacun peut exécuter du code écrit dans l'autre langage
  - Chacun peut étendre une classe écrite dans l'autre langage

- Mais des limitations dans le sens Scala → Java
  - Certains concepts de Scala n'existent pas en Java (e.g. traits, implicits)
  - Les structures de données doivent être converties avec la méthode asJava()

# JetBrains Educational Tools

- Validez automatiquement vos solutions (en cliquant sur « Check »)
- Laissez-vous guider pas à pas
  - Enoncé associé au fichier de code ouvert (ou à une étape de son édition)
  - Option « Hint » sur la plupart des énoncés
  - Restriction de la zone modifiable dans le fichier de code
  - Quelques mini-QCM
- Comparez votre solution avec la correction
- Visualisez votre état d'avancement, les exercices complétés
- Télécharge et installe automatiquement les bibliothèques externes
- Et bien sûr: auto-complétion, détection des erreurs de types, etc…

# Exercices

- Stream API
- Récursivité et optimisation
  - Mémoization
  - Récursivité terminale
- Implicites
- Programmation multi-cœur
- Spark
- Property-based testing

(vendredi)

| Scala | Java |
|---|---|
| ✓ | ✓ |
| ✓ | ✓ |
| ✓ | ✓ |
| ✓ | ✓ |
| ✓ | |
| ✓ | ✓ |
| ✓ | |
| ✓ | (✓) |