Introduction to Scala

MAIF

```scala
var param1: String = "param1"
val param2 = "param2"

def doSomething(param: String = "default"): Unit = {
  println(s"value = $param")
}
def doSomething(param: String = "default") {
  println(s"value = $param")
}

def process(s: String): String = s.toLowerCase

list.foreach { item =>
  println(item)
  println(item.toUpperCase)
}
list.foreach(item => item.toLowerCase)
list.foreach(_.toLowerCase)
list.foreach(process)
```

Basics

```
val name = "World"

println(s"hello $name!")

println(s"hello ${name.toUpperCase}!")
```

String interpolation

```scala
trait AuthService {

  def isConnected(username: String): Boolean

  def hash(username: String) = MD5.sign(username)
}

class MyService extends AuthService {
  override def isConnected(username: String): Boolean = true
}
```

Traits

```scala
class Post(id: String, title: String) {

  val innerContext = "Nouveau post »

  def display() = s"$title \n $innerContext"
}
```

Classes

```scala
case class User(id: String, name: String, email: String, age: Int)

object User {
  def findByEmail(email: String) = ???
  def birthday(u: User) = u.copy(age = u.age + 1)
}

object UserController {
  def getUrl(email: String) = User.findByEmail(email)
}
```

Case class

```scala
def doSomething(param: String) {
  println(s"value = $param")
}

def process(s: String): String = s.toLowerCase

list.foreach { item =>
  item.toUpperCase
}

list.foreach(_.toLowerCase)
list.foreach(process)

def doSomethingElse(param: String): String = {
  s"value = $param"
}
def doSomethingElse(param: String): String = s"value = $param"
```

Fonctions

```scala
object Hello {
  def apply(name: String) = println(s"Hello $name!")

  def doSomething(f: String => String) = f("Hello")
  def doSomething(f: (String, Int) => String) = f("Hello")

  def /?\(name: String) = apply(name)
}

Hello("World")
Hello.doSomething(p => p.toUpperCase)
Hello.doSomething { p =>
  p.toLowerCase
}

Hello./?\("World")
Hello /?\ "World"
```

Fonctions

```scala
def add(v1: Int)(v2: Int) = v1 + v2

val value1 = add(2)(2) // 4

val add4 = add(4)_

val val2 = add4(2) // 6
```

Curryfication

```scala
val param = ???

param match {
  case "Hello"                                                  => println("Yeah !")
  case 1                                                        => println("Yeah 1")
  case t: Task                                                  => println(t)
  case t: Task if t.id == 1                                     => println(t)
  case "a" :: "b" :: tail                                       => println(tail)
  case head :: "b" :: "c" :: tail                               => println(head)
  case User(id, name, email, age)                               => println(s"$id : $name : $email : $age")
  case User(_, _, _, age) if age > 18                           => println(age)
  case user @ User(_, _, _, age) if age > 18                    => println(user)
  case p @ Post(_, _, User(_, _, _, age)) if age < 100          => println(p)
}

Hello.doSomething {
  case "Hello" => println("Hello World")
  case _ => println("Yeah !")
}
```

Pattern matching

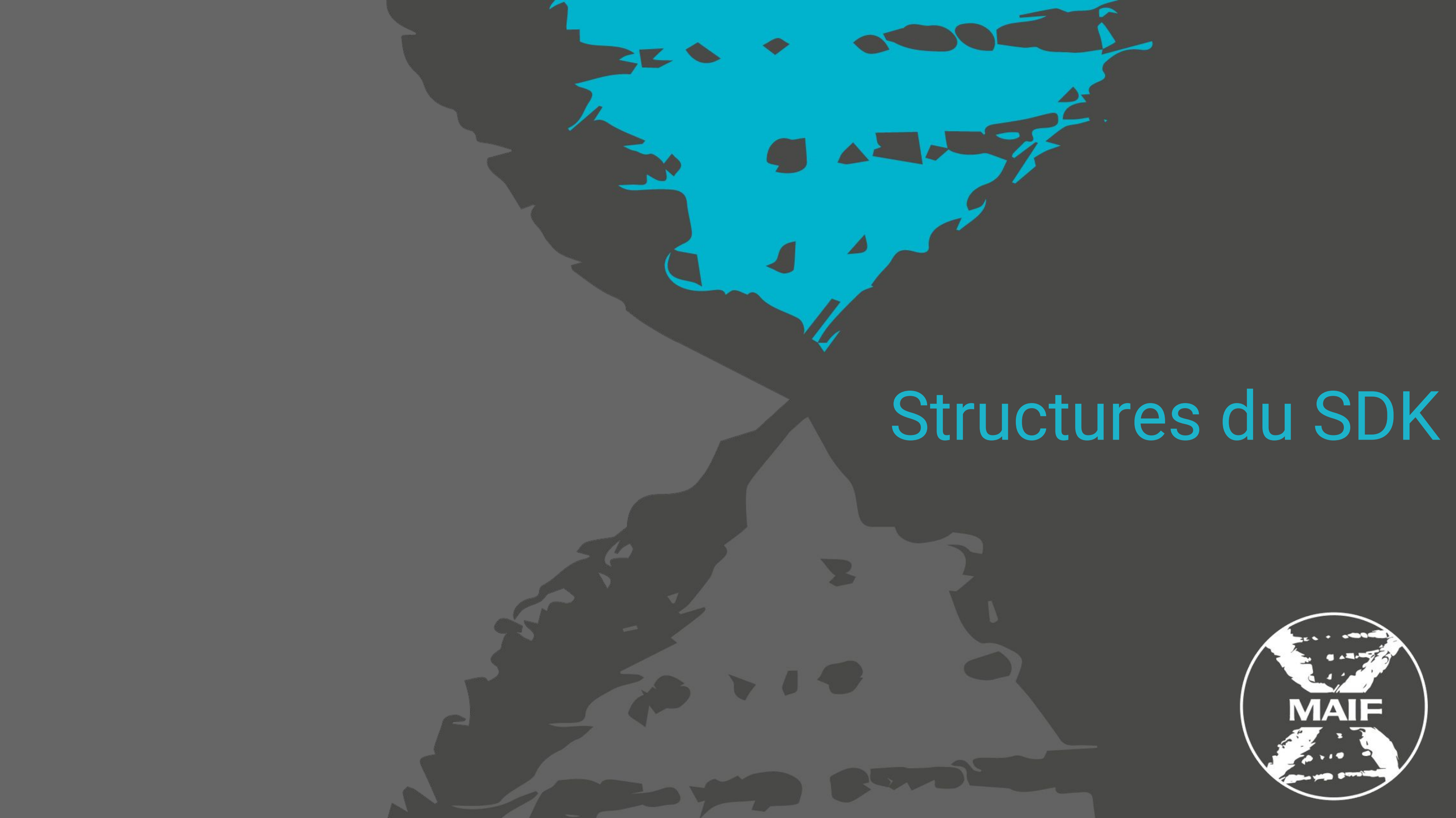
```scala
def adder(a: Int)(implicit b: Int) = a + b

adder(2)(2) // 4

implicit val implicitParam = 6

adder(2) // 8
```

Implicits



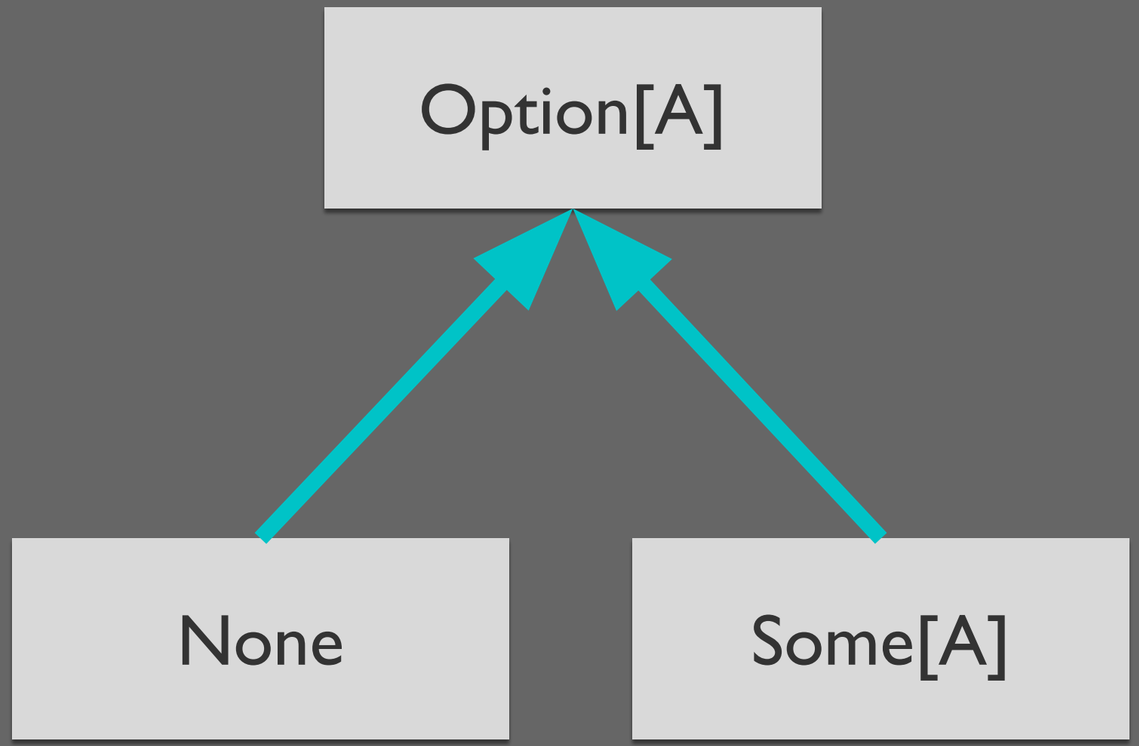
Structures du SDK

MAIF

```scala
val coll: Seq[String] =
    Seq("a", "b", "c", "d", "e", "f", "g")

val str: String = coll
      .filter(_.length == 1)
      .map(_.toUpperCase)
      .reduce(_ + _)    // ABCDEFG
```

Collections



Option

Option[A] map = (f: A => B): Option[B]

Option[A] flatMap = (f: A => Option[B]): Option[B]

Option[A] filter = (f: A => Boolean): Option[A]

Option

```
process(Option("Hello"))
process(None)

def process(opt: Option[String]) = {
  opt match {
    case Some(str) => println(str)
    case None => println("No value here")
  }
  opt.map(_.toUpperCase)
     .filter(_.length >= 5)
     .map(_ + "World")
     .getOrElse(":(") // "HELLO World"
}
```

Option

```scala
import scala.util._

val proc1 = Try(process(Option("yeah")))
val proc2 = Try {
  throw new RuntimeException("Error !!!!")
}
def handleTry(t: Try[Option[String]]) = {
  t match {
    case Success(Some(str)) => println(str)
    case Success(None) => println("Success but nothing found")
    case Failure(err) => println(err.getMessage)
  }
}

handleTry(proc1)
handleTry(proc2)
```

Try

```scala
implicit val ec = ExecutionContext.fromExecutor(Executors.newFixedThreadPool(5))

val future1 = Future {
  Thread.sleep(10000)
  "I'm done 1!"
}
val future2 = Future {
  Thread.sleep(20000)
  "I'm done 2!"
}

future1.map { message =>
  println(message)
  message
}.onComplete {
  case Success(message) => println(s"success of : $message")
  case Failure(err) => println(s"error : ${err.getMessage}")
}
```

Future

```scala
implicit val ec = ExecutionContext.fromExecutor(Executors.newFixedThreadPool(5))

val future1 = Future {
  Thread.sleep(10000)
  "I'm done 1! »
}
val future2 = Future {
  Thread.sleep(20000)
  "I'm done 2!"
}

future1.flatMap { message1 =>
  future2.map { message2 =>
    s"messages : $message1, $message2"
  }
}.onComplete {
  case Success(message) => println(message) // messages: I'm done 1!, I'm done 2!
  case Failure(err) => println(err.getMessage)
}(ec)
```

Future