De Java a Scala:
cómo conocí la programación funcional

If you have a dream, we can write the code

intelygenz

About me:

DarkRodry

bq

tuenti

intelygenz

t3f

Betabeers

If you have a dream, we can write the code

If you have a dream, we can write the code
intelygenz

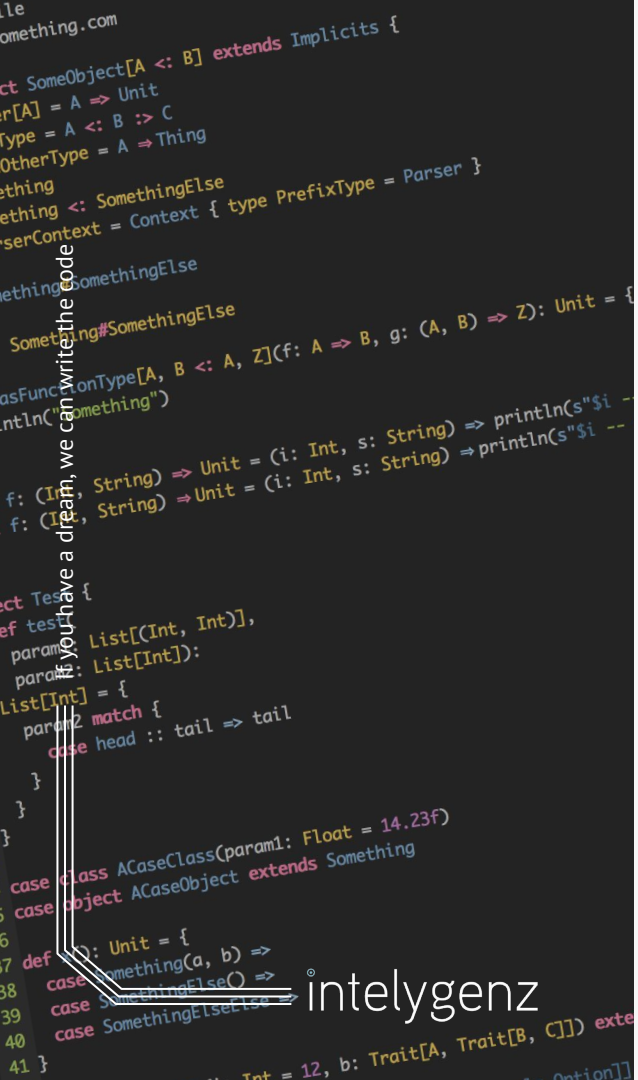
If you have a dream, we can write the code
intelygenz

If you have a dream, we can write the code
WESLEYAN
intelygenz

If you have a dream, we can write the code

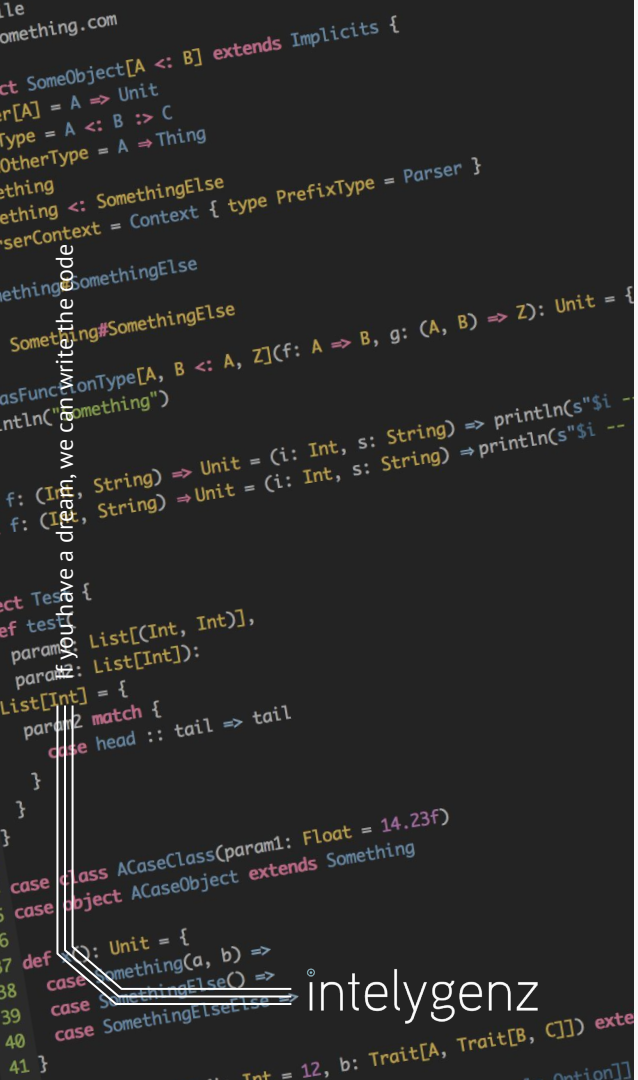
intelygenz

If you have a dream, we can write the code
intelygenz



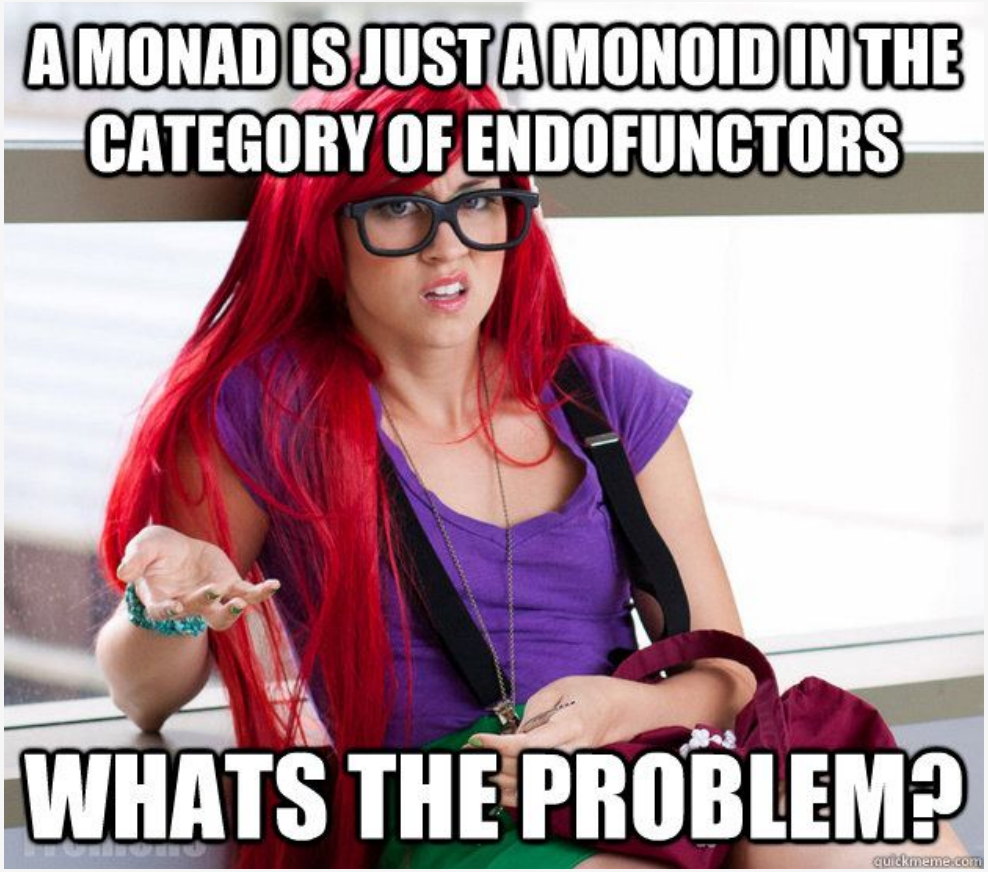
Scala

- Orientado a objetos
- Funcional
- Compila a Java bytecode
  - funciona sobre la JVM



Scala

```scala
object HelloWorld {
    def main(args: Array[String]) {
        println("Hello t3chFest!")
    }
}
```


A MONAD IS JUST A MONOID IN THE CATEGORY OF ENDOFUNCTORS
WHATS THE PROBLEM?
quickmeme.com

```java
boolean b = false;
int i = 1;
float f = 3.5f;
char c = 'J';
String  s = "hello world";
List <String> list = new ArrayList<>();
```

```scala
var b: Boolean = true
var i: Int = 1
var f: Float = 3.5f
var c: Char = 'S'
var s: String = "hello world"
var s: List[String] = List("hello",
"world")
```

If you have a dream, we can write the code

```java
final boolean b = false;
final int i = 1;
final float f = 3.5f;
final char c = 'J';
final String  s = "hello world";
List <String> list = new
ArrayList<>();
list.add("hello");
list.add("world");
```

```scala
val b: Boolean = true
val i: Int = 1
val f: Float = 3.5f
val c: Char = 'S'
val s: String = "hello world"
val s: List[String] = List("hello",
"world")
```

If you have a dream, we can write the code

intelygenz

```
public class Suit {

    final String color;
    final long size;

    public Suit(String color, long
size) {
        this.color = color;
        this.size = size;
    }
}
```

```
case class Suit(color: String, size:
Long)
```

If you have a dream, we can write the code

```java
public String suitUp(Suit suit) {
    return "My " + suit.color + "
suit!";
}
```

```scala
def suitUp(suit: Suit): String = {
   s"My legendary ${suit.color} suit"
}

val f: Suit => String = suitUp

def sayIt(a: String)(b: String): String = {
    s"$a$b"
}

val g: String => String => String = sayIt
val curry: String => String =
sayIt("legen")
val quote: String = curry("dary")
```

If you have a dream, we can write the code

If you have a dream, we can write the code

```
sealed trait Clothes
case class Suit(color: String, size: Long) extends Clothes
case class Sweater(thickness: Int) extends Clothes
case class TShirt(color: String, quote: String) extends Clothes

def suitUp(clothes: Clothes): String = clothes match {
    case Suit(color, _) => s"My legendary $color suit!"
    case Sweater(_, _) => "Is it your grandmother's?"
    case _ => "This clothes sucks"
}
```

```java
public interface Awesomeness {
    String highFive();
    String playLaserTag();
    boolean flirt(Girl girl);
}
```

```scala
trait Awesomeness {
    def highFive(): String
    def playLaserTag(): String
    def flirt(girl: Girl): Boolean
}
```

If you have a dream, we can write the code

If you have a dream, we can write the code

```scala
trait BroCode[T] {
    def suitUp(x: T): T
    def drink(x: T, d: Drink): Drink
    def acceptChallenge(x: T, f: T => T): T
}
```

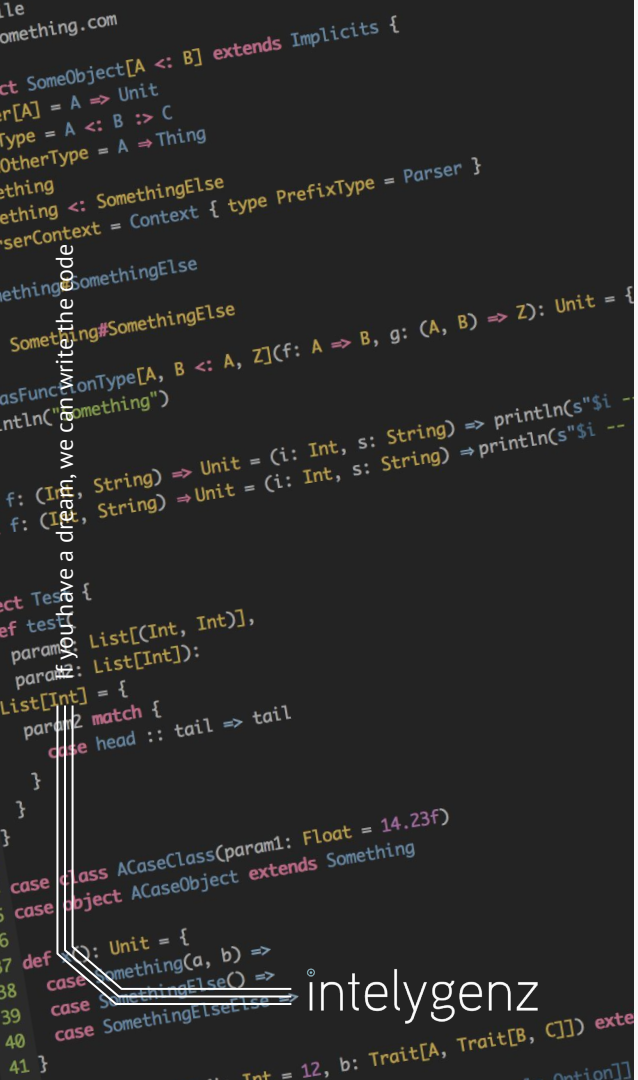
```scala
implicit object TedBroCode extends
  BroCode[Ted] {
    def suitUp(x: Ted): Ted =
        ElegantTed()
    def drink(x: Ted, y: Drink): Drink =
        EmptyGlass()
    def acceptChallenge(x: Ted,
        f: Ted => Ted): Ted = f(x)
}
```

```scala
implicit object BarneyBrocode extends
  BroCode[Barney] {
    def suitUp(x: Barney): Barney = x
    def drink(x: Barney, d: Drink): Drink =
        d match {
            case Water() => d
            case _ => (x, EmptyGlass())
        }
    def acceptChallenge(x: Barney,
        f: Barney => Barney): Barney = f(x)
}
```

If you have a dream, we can write the code

```scala
def goToMaclarens[T: BroCode](t: T): T =
    for {
        tSuited <- suitUp(t)
        _ <- drink(tSuited, Beer())
        tFinish <- acceptChallenge(tSuited, playLasertag(x))
    } yield tFinish
```

```scala
val normalDay: Barney = goToMaclarens[Barney](barney)
val randomDay: Ted = goToMaclarens[Ted](ted)
```



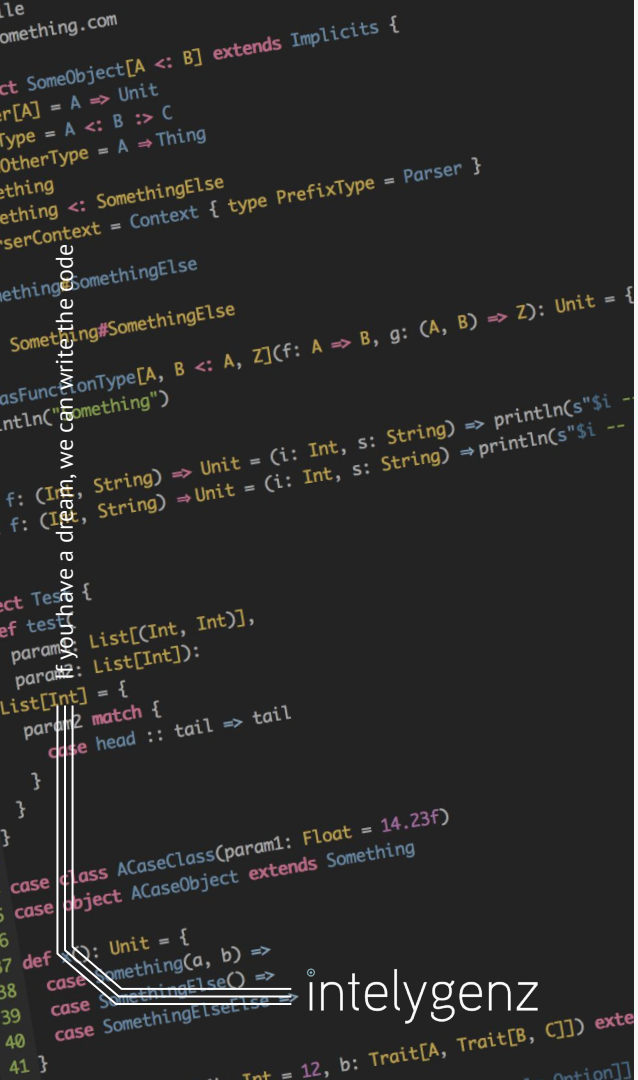
If you have a dream, we can write the code

intelygenz

scalaz

If you have a dream, we can write the code
intelygenz

If you have a dream, we can write the code



Enlaces de interés

- *Functional Programming in Scala,* Paul Chiusano y Rúnar Bjarnason
  http://amzn.to/2kwXpkj
- *Principios de Programación Funcional en Scala*, Coursera
  https://www.coursera.org/learn/progfun1
- *Scala Exercises*, 47 Degrees https://www.scala-exercises.org/
- *The Neophyte's Guide to Scala*. Daniel Westheide
  http://danielwestheide.com/scala/neophytes.html
- *ScalaMAD Meetup* https://www.meetup.com/Scala-Programming-Madrid