/\*\*

\* A person has a name and an age.

\*/

case class Person(name: String, age: Int)

abstract class Vertical extends CaseJeu

case class Haut(a: Int) extends Vertical

case class Bas(name: String, b: Double) extends Vertical

sealed trait Ior[+A, +B]

case class Left[A](a: A) extends Ior[A, Nothing]

case class Right[B](b: B) extends Ior[Nothing, B]

case class Both[A, B](a: A, b: B) extends Ior[A, B]

trait Functor[F[\_]] {

def map[A, B](fa: F[A], f: A => B): F[B]

}

// beware Int.MinValue

def absoluteValue(n: Int): Int =

if (n < 0) -n else n

def interp(n: Int): String =

s"there are $n ${color} balloons.\n"

type ξ[A] = (A, A)

trait Hist { lhs =>

def ⊕(rhs: Hist): Hist

}

def gsum[A: Ring](as: Seq[A]): A =

as.foldLeft(Ring[A].zero)(\_ + \_)

val actions: List[Symbol] =

'init :: 'read :: 'write :: 'close :: Nil

trait Cake {

type T;

type Q

val things: Seq[T]

abstract class Spindler

def spindle(s: Spindler, ts: Seq[T], reversed: Boolean = false): Seq[Q]

}

val colors = Map(

"red" -> 0xFF0000,

"turquoise" -> 0x00FFFF,

"black" -> 0x000000,

"orange" -> 0xFF8040,

"brown" -> 0x804000)

lazy val ns = for {

x <- 0 until 100

y <- 0 until 100

} yield (x + y) \* 33.33