IO Monad - Scalaz

https://apocalisp.wordpress.com/2011/12/19/towards-an-effect-system-in-scala-part-2-io-monad/

Runar Bjarnason

```
1 type ST[S, A] = World[S] => (World[S], A)
```

The IO data type is very similar, except that we fix the world-state to be of a specific type:

```
1 type IO[A] = ST[RealWorld, A]
```

```
1 def program: IO[Unit] = for {
2 line <- readLn
3      <- putStrLn(line)
4 } yield ()
```

Direct conversion from Haskell

IO Monad - Monix (2.x series)

https://monix.io/docs/2x/eval/task.html#design-summary

Alex Nedelcu

```
import monix.
import scala.util.Random
import monix.execution.Scheduler.Implicits.global

val t1 = {
  val r = new Random(0L)
  val x = Task(r.nextInt)
  for {
    a <- x
    b <- x
  } yield (a, b)
}

// Same as f1, but I inlined `x`
val t2 = {
  val r = new Random(0L)
  for {
    a <- Task(r.nextInt)
    b <- Task(r.nextInt)
  } yield (a, b)
}
```

Now you'll find that both t1 and t2 return the same value `(-1155484576,-723955400)`