

IO Monad – Future

```
import scala.concurrent.future
import scala.util.Random
import scala.concurrent.ExecutionContext.Implicits.global

val f1 = {
  val r = new Random(0L)
  val x = Future(r.nextInt)
  for {
    a <- x
    b <- x
  } yield (a, b)
}

// Same as f1, but I inlined `x`
val f2 = {
  val r = new Random(0L)
  for {
    a <- Future(r.nextInt)
    b <- Future(r.nextInt)
  } yield (a, b)
}
```

IO Monad – Future

Future is not a Monad

Future execution is eager

<http://justinhj.github.io/2018/05/05/hacker-news-api-4.html>

```
import scala.concurrent.future
import scala.util.Random
import scala.concurrent.ExecutionContext.Implicits.global
```

```
val f1 = {
  val r = new Random(0L)
  val x = Future(r.nextInt)
  for {
    a <- x
    b <- x
  } yield (a, b)
}
```

```
// Same as f1, but I in
val f2 = {
  val r = new Random(0L)
  for {
    a <- Future(r.nextInt)
    b <- Future(r.nextInt)
  } yield (a, b)
}
```

In this example, we are running some side-effecting code in the Future (generating a random number mutates the Random object by updating its seed). The result of running f1 is:

```
Future[(Int, Int)] = Future(Success((-1155484576,-1155484576)))
```

Whilst f2 gives:

```
Future[(Int, Int)] = Future(Success((-1155484576,-723955400)))
```

For referential transparency, we can take any function and its arguments and replace it with the result.

```
val x = something
(x, x)
```

should be the same as

```
(something, something)
```