Monads

Monads

Higher-kinded type class that provides

- a pure method to wrap a normal value into a monadic value
- a flatMap method to transform monadic values in sequence

```
import cats.Monad
import cats.instances.list._

val optionMonad = Monad[Option] // fetches the implicit instance
val anOption = optionMonad.pure(2) // returns an Option[Int]
val aTransformedOption = optionMonad.flatMap(anOption) { x =>
  if (x % 2 == 0) Some (x + 1) else None
}
```

Can implement map in terms of pure + flatMap

- Monads extend Functors

Monads

Extension methods are in other packages

- will become clearer by the end of the course

```
import cats.syntax.applicative._ // adds the pure extension method
val oneValid = 1.pure[ErrorOr]
```

```
import cats.syntax.functor._ // adds the map extension method
val twoValid = oneValid.map(_ + 1)
```

```
import cats.syntax.flatMap._ // adds the flatMap extension method
val transformedValue = oneValid.flatMap(x => (x + 1).pure[ErrorOr])
```

map + flatMap = for-comprehensions

```
val composedErrorOr = for {
  one <- oneValid
  two <- twoValid
} yield one + two
```

Monads

Use cases: sequential transformations

- list combinations
- option transformations
- asynchronous chained computations
- dependent computations

For-comprehensions are NOT ITERATION.

Step away from the concept of iteration.

FlatMap is a mental model of chained transformations.

```
def getPairs[M[_] : Monad, A, B](ma: M[A], mb: M[B]): M[(A, B)] = for {
  a <- ma
  b <- mb
} yield (a, b)
```

Cats rock