You know (some) category theory already

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[loop above, "1_A"] \arrow[dr, "g\circ f"'] & B \arrow[d, "g"] \arrow[loop above, "1_B"] \\
& C \arrow[loop below, "1_C"]
\end{tikzcd}$$

$$\begin{tikzcd}
X \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] & Y \arrow[r, "q"] \arrow[dr, "q'"'] & Q \arrow[d, dotted] \\
& & Q'
\end{tikzcd}$$

$$\begin{tikzcd}
& & & A_2 \arrow[rr] \arrow[dd, "a_2"] & & B_2 \arrow[rr] \arrow[dd, "b_2"] & & C_2 \arrow[rr] \arrow[dd, "c_2"] & & 0 \\
& & A_1 \arrow[ur] \arrow[rr] \arrow[dd, "a_1"] & & B_1 \arrow[ur] \arrow[rr] \arrow[dd, "b_1"] & & C_1 \arrow[ur] \arrow[rr] \arrow[dd, "c_1"] & & 0 & \\
& 0 \arrow[rr] & & A_2' \arrow[rr] & & B_2' \arrow[rr] & & C_2' & & \\
0 \arrow[rr] & & A_1' \arrow[ur] \arrow[rr] & & B_1' \arrow[ur] \arrow[rr] & & C_1' \arrow[ur] & & &
\end{tikzcd}$$

Category Theory

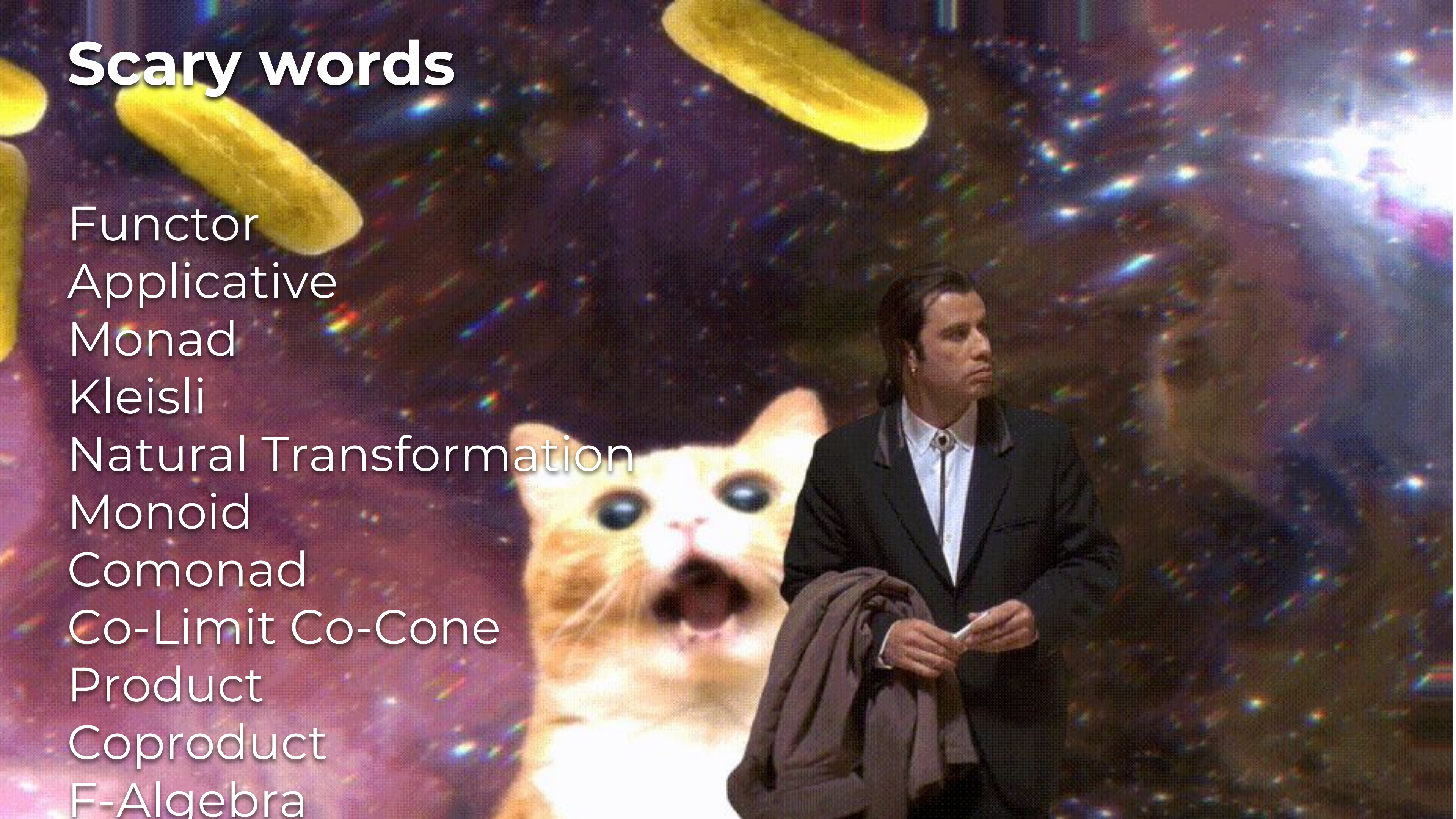
Most abstract flavour of maths

Describes **behaviour** of instead of implementation

Types and **functions** form a category

Theory behind **Cats/ScalaZ**

Scary words



Functor
Applicative
Monad
Kleisli
Natural Transformation
Monoid
Comonad
Co-Limit Co-Cone
Product
Coproduct
F-Algebra

Scary words

Functor

you already know!

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

 List[T]

 Future[T]

 Option[T]

 Either[A, T]

 Validated[E, T]

 (A, T)
```

Scary words you already know!

Functor laws

1. Identity

```
List(1,2,3).map(x => x) ==
List(1,2,3)
```

2. Associativity

```
Option("fun")
    .map(_ + "ct")
    .map(_ + "or") ==
Option("fun")
    .map(_ + "ct" + "or")
```

Unfamiliar things

Applicative

```
extends Functor
```

It's fine…

```
List[T]

Future[T]

Option[T]

Either[A, T]

Validated[E, T]
```

Unfamiliar things

Applicative

```
extends Functor
```

```
x.pure[A]
```

It’s fine...

```
List(1)

Future.successful("yes")

Some(1)

Right(2)

Validated.valid("easy")
```

Unfamiliar things

Applicative

```
extends Functor
```

```
(a map2 b)(_ + _)
```

It’s fine…

“Parallel computations”

“zip and then map”

```
(List(1, 2, 3) zip List(3, 2, 1)).map {
    case (a, b) => a + b
}
```

Just a name

Monad

```
extends Applicative
```

No problem

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

```
List[T]
```

```
Future[T]
```

```
Option[T]
```

```
Either[A, T]
```

“Sequential computations”

“Dependent computations”

Unknown territory

Kleisli

That is familiar

```
A => M[B]
```

“Argument of a flatMap”

```
Future.successful(5).flatMap{
 x => Future.successful(x + 1)
}
```

Unknown territory

Kleisli

“Monadic function”

```
f: A => M[B]
g: B => M[C]

f andThen g : A => M[C]

A => M[B]
A => M[M[C]]    using g
A => M[C]
```

That is familiar

```
A => M[B]
```

“Argument of a flatMap”

```
Future.successful(5).flatMap{
 x => Future.successful(x + 1)
}
```

Long words

Natural transformation

FunctionK

```
F ~> G
```

We use constantly

```
F[X] => G[X]

List(1, 2, 3).headOption

Option(5).toList

Await.result(future, 5 seconds)
```

Abstract stuff

Monoid `T` `f`

Combine

```
f: (T,T) => T
```

Associativity

```
f(a, f(b,c)) ==
f(f(a,b),c)
```

Identity element

```
T
```

We use constantly

```
Int +
```

“Brackets don’t matter”

```
5 + 6 + 7
(5 + 6) + 7
5 + (6 + 7)
```

“+0 does nothing”

```
1 + 0 == 1 == 0 + 1
```

Category theory yo

“Monad is a monoid in the category of endofunctors”

Category theory yo

“Monad is a monoid in the category of ~~endo~~functors”

Category theory yo

“Monad is a monoid in the category of ~~endo~~functors”

Combine

```
(T,T) => T
```

Identity

```
T
```

Category theory yo

“Monad is a monoid in the category of ~~endo~~functors”

Combine

```
(T,T) => T       F[F[X]] => F[X]
```

Identity

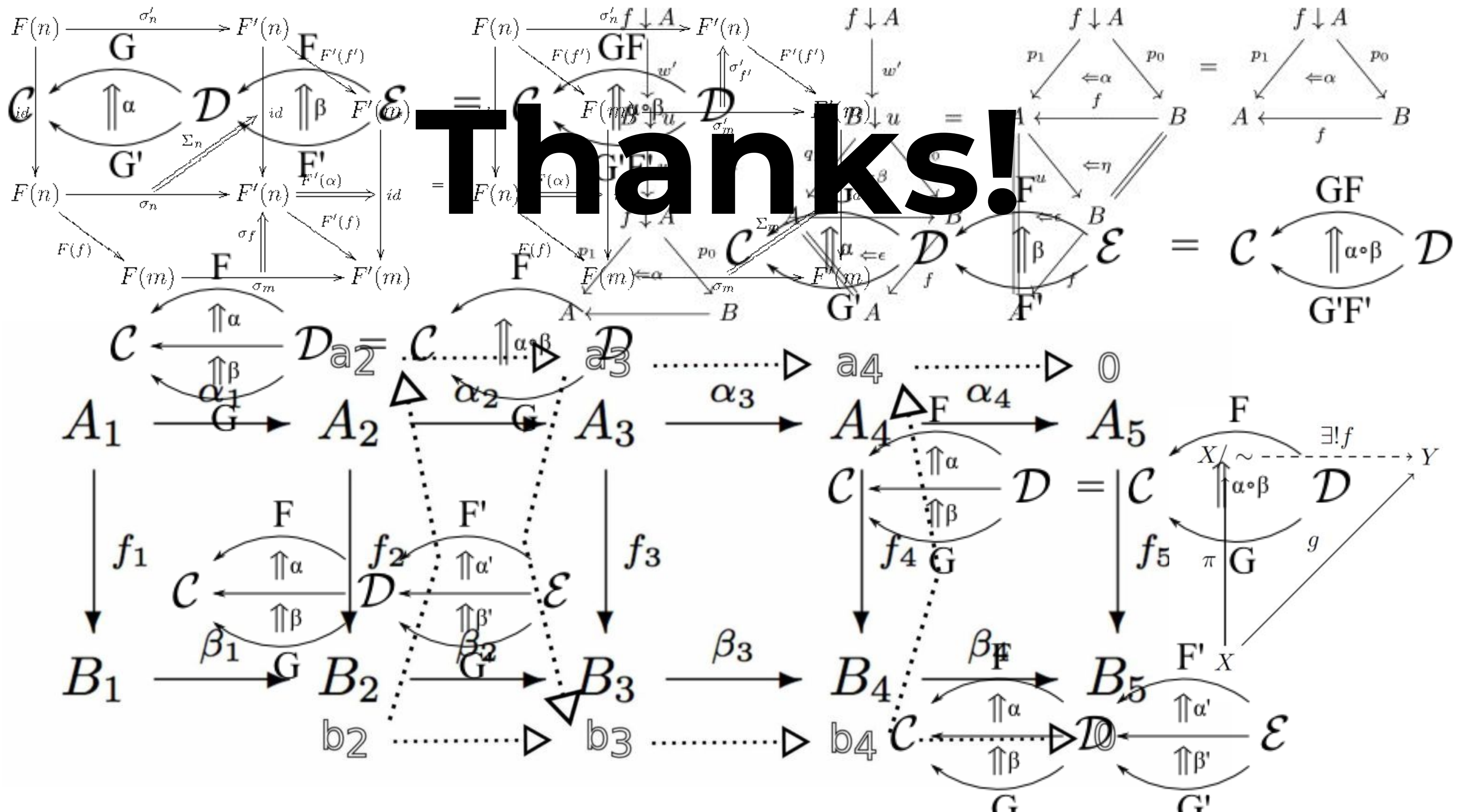
```
T                F[X]
```

Category theory yo

“Monad is a monoid in the category of ~~endo~~functors”

Combine `(T,T) => T`	“A way of flattening” `F[F[X]] => F[X]`	`flatMap`
Identity `T`	“A way of wrapping” `F[X]`	`pure`

Thanks!



A_1
A_2
A_3
A_4
A_5
B_1
B_2
B_3
B_4
B_5
α_1
α_2
α_3
α_4
β_1
β_2
β_3
β_4
f_1
f_2
f_3
f_4
f_5
a2
a3
a4
0
b2
b3
b4
$\mathcal{C}$
$\mathcal{D}$
$\mathcal{E}$
F
G
F'
G'
GF
G'F'
$\alpha\circ\beta$
$F(n)$
$F'(n)$
$F(m)$
$F'(m)$
$f \downarrow A$
p_1
p_0
$X/\sim$
Y
X
$\exists! f$
g
π