# Rockin' in a free world

Alessandro Candolini

November 19, 2022

# Agenda

1. Free monoids
2. Free monads
3. Polysemy
4. Free monoidals
5. Optparse

# Free monoids

# Free monads

# Placeholder Example

# Polysemy

# Free monoidals

**Definition (monoid)**

A "monoid" is a tuple $(A, \varphi, e)$ where

- A is a set[1]
- $\varphi : A \times A \to A$ is an *associative* binary operation on A
- $e \in A$ is a "neutral" element, ie, for every $a \in A$, $\varphi(a, e) = \varphi(e, a) = a$

[1] For all practical purposes, it's convenient to restrict the definition to non-empty sets.

```
class Monoid a where
    (<>) :: a -> a -> a -- binary operation
    mempty :: a  -- neutral element
```

```
class Semigroup a where
  (<>) :: a -> a -> a -- binary operation

class Semigroup a => Monoid a where
  mempty :: a -- neutral element
```

Monoids are everywhere:

- $(\mathbb{N}, +, 0)$ is a (commutative) monoid
- $(\mathbb{N}, \times, 1)$ is a (commutative) monoid
- String concatenation is a (non-commutative) monoid, with empty string as neutral element
- etc

```haskell
{-# LANGUAGE DerivingVia #-}

newtype AdditiveInteger = AdditiveInteger Integer
  deriving (Eq,Show)
  deriving Num via Integer

instance Monoid AdditiveInteger where
   (<>) = (+)
   mempty = 0
```

```haskell
{-# LANGUAGE DerivingVia #-}

newtype MultiplicativeInteger = MultiplicativeInteger
  deriving (Eq,Show)
  deriving Num via Integer

instance Monoid MultiplicativeInteger where
   (<>) = (*)
   mempty = 1
```

```haskell
{-# LANGUAGE FlexibleInstances #-}

instance Monoid String where
    (<>) = (++)
    mempty = ""
```

Monoids, categorically

# Optparse

# Questions?