Purescript - Prelude

1 Monoid

```
class (Eq a) <= Ord a where
    compare :: a -> a -> Ordering

(<>) :: Semigroup a => a -> a -> a
```

2 Foldable

```
class Foldable t where
  foldMap :: Monoid m => (a -> m) -> t a -> m
  foldr   :: (a -> b -> b) -> b -> t a -> b

traverse_ :: (a -> f b) -> t a -> f ()
for_      ::  t a -> (a -> f b) -> f ()
sequenceA_:: t (f a) -> f ()

concat    :: t [a] -> [a]
and       :: t Bool -> Bool
or        :: t Bool -> Bool
any       :: (a -> Bool) -> t a -> Bool
all       :: (a -> Bool) -> t a -> Bool
```

t :: Foldable, f :: Applicative

3 Functor

```
class Functor f where
  fmap :: (a -> b) -> f a -> f b

(<$)   :: a -> f b -> f a
($>)   :: f a -> b -> f b
(<$>)  :: (a -> b) -> f a -> f b
(<&>)  :: f a -> (a -> b) -> f b
void   :: f a -> f ()
```

f :: Functor

4 Applicative

```
class Functor f => Applicative f where
  pure  :: a -> f a
  (<*>) :: f (a -> b) -> f a -> f b
  (*>)  :: f a -> f b -> f b
  (<*)  :: f a -> f b -> f a
  liftA2:: (a -> b -> c) -> f a -> f b -> f c

(<**>)  :: f a -> f (a -> b) -> f b

liftA   :: (a -> b) -> f a -> f b
liftA3  :: (a -> b -> c -> d) -> f a -> f b -> f c -> f d

void    :: f a -> f ()
forever :: f a -> f b
when    :: Bool -> f () -> f ()
unless  :: Bool -> f () -> f ()
```

f :: Applicative

5 Alternative

```
class Applicative f => Alternative f where
  empty :: f a
  (<|>) :: f a -> f a -> f a
  some  :: f a -> f [a]
  many  :: f a -> f [a]

optional :: f a -> f (Maybe a)
guard    :: Bool -> f ()
```

f :: Alternative

6 Traversable

```
class (Functor t, Foldable t) => Traversable t where
  traverse  :: (a -> f b) -> t a -> f (t b)
  sequenceA :: t (f a) -> f (t a)
  mapM      :: (a -> m b) -> t a -> m (t b)
  sequence  :: t (m a) -> m (t a)

for :: t a -> (a -> f b) -> f (t b)
```

f :: Applicative, t :: Traversable, m :: Monad

7 Monad

```
class Applicative m => Monad m where
  (>>=)  :: forall a b. m a -> (a -> m b) -> m b
  (>>)   :: forall a b. m a -> m b -> m b
  return :: a -> m a

mapM_       :: (a -> m b) -> t a -> m ()
forM        :: t a -> (a -> m b) -> m (t b)
forM_       :: t a -> (a -> m b) -> m ()
sequence_   :: t (m a) -> m ()

(=<<)       :: (a -> m b) -> m a -> m b
(>=>)       :: (a -> m b) -> (b -> m c) -> a -> m c
(<=<)       :: (b -> m c) -> (a -> m b) -> a -> m c

join        :: m (m a) -> m a
filterM     :: (a -> m Bool) -> [a] -> m [a]
filterM     :: (a -> m Bool) -> [a] -> m [a]
foldM       :: (b -> a -> m b) -> b -> t a -> m b
foldM_      :: (b -> a -> m b) -> b -> t a -> m ()
replicateM  :: Int -> m a -> m [a]
replicateM_ :: Int -> m a -> m ()


liftM       :: (a1 -> r) -> m a1 -> m r
liftM2      :: (a1 -> a2 -> r) -> m a1 -> m a2 -> m r
ap          :: m (a -> b) -> m a -> m b
(<$!>)      :: (a -> b) -> m a -> m b
```

f :: Applicative, t :: Traversable, m :: Monad