How to print a tree

```haskell
data Tree a = Leaf a | Branch [Tree a]
```

How to print a tree

```haskell
data Tree a = Leaf a | Branch [Tree a]
```

How to print a tree

```
data Tree a = Leaf a | Branch [Tree a]
  deriving (Show)
```

How to print a tree

```haskell
data Tree a = Leaf a | Branch [Tree a]
```

```haskell
instance (Show a) => Show (Tree a) where
  show :: Tree a -> String
  show (Leaf x) = "Leaf (" ++ show x ++ ")"
  show (Branch ts) = "Branch " ++ show ts
```

How to print a tree

```haskell
{-# LANGUAGE StandaloneDeriving #-}

data Tree a = Leaf a | Branch [Tree a]

deriving instance (Show a) => Show (Tree a)
```

How to print a tree

```
{-# LANGUAGE StandaloneDeriving #-}

data Tree a = Leaf a | Branch [Tree a]
```

Many options:

```
  deriving (Show)

instance (Show a) => Show (Tree a) where
  show :: Tree a -> String
  show (Leaf x) = "Leaf (" ++ show x ++ ")"
  show (Branch ts) = "Branch " ++ show ts

deriving instance (Show a) => Show (Tree a)
```

How to print a tree

```haskell
{-# LANGUAGE StandaloneDeriving #-}

data Tree a = Leaf a | Branch [Tree a]
```

Many options:

```haskell
  deriving (Show)
```

```haskell
instance (Show a) => Show (Tree a) where
  show :: Tree a -> String
  show (Leaf x) = "Leaf (" ++ show x ++ ")"
  show (Branch ts) = "Branch " ++ show ts
```

```haskell
deriving instance (Show a) => Show (Tree a)
```

Discussion

- Not all at once
  - `OverlappingInstances`
- See also

How to print a tree

```haskell
{-# LANGUAGE StandaloneDeriving #-}

data Tree a = Leaf a | Branch [Tree a]
```

Many options:

```haskell
  deriving (Show)
```

```haskell
instance (Show a) => Show (Tree a) where
  show :: Tree a -> String
  show (Leaf x) = "Leaf (" ++ show x ++ ")"
  show (Branch ts) = "Branch " ++ show ts
```

```haskell
deriving instance (Show a) => Show (Tree a)
```

Discussion

- Not all at once
  - `OverlappingInstances`
- See also

How to print a tree

```haskell
{-# LANGUAGE StandaloneDeriving #-}

data Tree a = Leaf a | Branch [Tree a]
```

Many options:

```haskell
  deriving (Show)
```

```haskell
instance (Show a) => Show (Tree a) where
  show :: Tree a -> String
  show (Leaf x) = "Leaf (" ++ show x ++ ")"
  show (Branch ts) = "Branch " ++ show ts
```

```haskell
deriving instance (Show a) => Show (Tree a)
```

Discussion

- Not all at once
  - `OverlappingInstances`
  - Choose max. 1!
- See also

How to print a tree

```haskell
{-# LANGUAGE StandaloneDeriving #-}

data Tree a = Leaf a | Branch [Tree a]
```

Many options:

```haskell
  deriving (Show)
```

```haskell
instance (Show a) => Show (Tree a) where
  show :: Tree a -> String
  show (Leaf x) = "Leaf (" ++ show x ++ ")"
  show (Branch ts) = "Branch " ++ show ts
```

```haskell
deriving instance (Show a) => Show (Tree a)
```

Discussion

- Not all at once
  - `OverlappingInstances`
- See also

How to print a tree

```haskell
{-# LANGUAGE StandaloneDeriving #-}

data Tree a = Leaf a | Branch [Tree a]
```

Many options:

```haskell
  deriving (Show)
```

```haskell
instance (Show a) => Show (Tree a) where
  show :: Tree a -> String
  show (Leaf x) = "Leaf (" ++ show x ++ ")"
  show (Branch ts) = "Branch " ++ show ts
```

```haskell
deriving instance (Show a) => Show (Tree a)
```

Discussion

- Not all at once
  - `OverlappingInstances`
- See also

How to print a tree

```haskell
{-# LANGUAGE StandaloneDeriving #-}

data Tree a = Leaf a | Branch [Tree a]
```

Many options:

```haskell
  deriving (Show)
```

```haskell
instance (Show a) => Show (Tree a) where
  show :: Tree a -> String
  show (Leaf x) = "Leaf (" ++ show x ++ ")"
  show (Branch ts) = "Branch " ++ show ts
```

```haskell
deriving instance (Show a) => Show (Tree a)
```

Discussion

- Not all at once
  - `OverlappingInstances`
- See also
  - Tweag on run-time instances