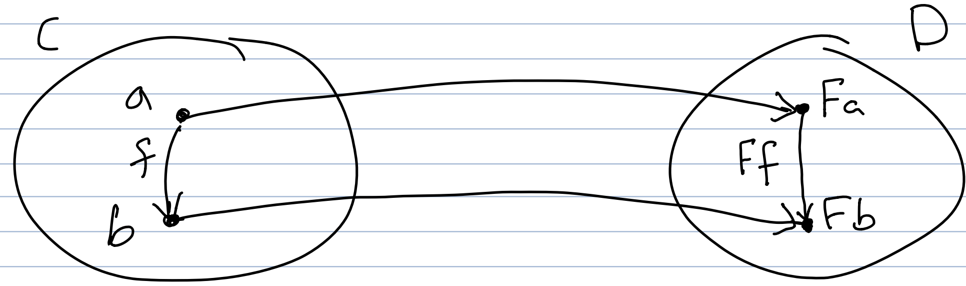
A **functor** is a mapping from one category to another.

A functor, F, preserves structure,



Specifically, it maps the hom-set from category C to the hom-set of D

$$C(a,b) \longrightarrow D(Fa, Fb)$$

Structure preservation specifically means preserving composition and identity.

$$F(g \circ f) = Fg \circ Ff \qquad \text{composition}$$

$$F(id_a) = id_{Fa} \qquad \text{identity}$$

Notice a functor does not need to be injective or surjective. It only needs to preserve the structure that it does map.

A **faithful** functor is <u>injective</u> on hom-sets
A **full** functor is <u>surjective</u> on hom-sets

An example in Haskell

To define a functor that maps a type to a Maybe type.

$$\begin{tikzcd} a \arrow[r] \arrow[d, "f"'] & \text{Maybe } a \arrow[d, "\text{fmap } f"] \\ b \arrow[r] & \text{Maybe } b \end{tikzcd}$$

```
fmap :: (a -> b) -> (Maybe a -> Maybe b)
```

fmap takes a function from a to b and returns a function from Maybe a to Maybe b.
fmap can be defined as:

```
fmap f Nothing = Nothing
fmap f Just x  = Just (f x)
```

To prove this is a functor, we must show it preserves identity and composition.

The following method of proof is called **equational reasoning**.

To prove identity is preserved, consider the 2 cases: Nothing and Just. Show the two sides of the function definition can match eachother.

```
fmap id Just x  = Just (id x)
                = Just x
                = id Just x
```

(using the def. of id x = x)

The Nothing case is equally trivial to prove.

The following statement on composition:

$$\text{fmap}\,(g \circ f) = \text{fmap}\, g \circ \text{fmap}\, f$$

is a **free theorem** because this is a polymorphic function that preserves identity.

The fmap function as defined in Haskell

```
fmap :: (a -> b) -> f a -> f b
```

utilizes **adhoc polymorphism**. That is, a specific implementation of fmap is defined for every type it is used for.

This is achieved by defining a **typeclass**, a family of types that adhere to the class interface.

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

Haskell can actually infer that f is a type constructor, not a type itself.