A **category** consists of   
- **Objects** as a collection or   
- **Morphisms** or **arrows** as a function:   
- A **composition**: (We write )  
- An **identity**:   
satisfying the following properties:  
- **Composition is associative**:   
, .  
- **Composition unit laws**: and   
Intuition. A category is a collection of objects and arrows between them, where it is possible to compose arrows that follow each other (composition priority doesn’t matter), and where each object has a special identity arrow from and to itself that, when composed with any other arrow, doesn’t change it.

We define **Hom**  
The **source function** of a category is    
The **target function** of a category is

A pair of arrows in a category is an **isomorphism** if , , and .

A **preorder** **relation** on a class consists of a binary relation on (where we say has lower or equal price to if ) such that in :  
- each thing has lower or equal price to itself  
- if one thing has lower or equal price to a second thing which has lower or equal price to a third thing, then the first has lower or equal price to the third thing.

A **preorder** relation on a class gives a category when:   
- we regard elements as the objects.  
- an arrow represents the fact that two elements are (with that choice, there is either 0 or 1 arrow between objects, so composition and identity are completely determined).

is a group iff its law is associative, there is a unit element, each element has an inverse.  
 is a monoid iff its law is associative, there is a unit element.

The **delooping of a monoid**  denoted is the following category:  
There is a single object :  •  
Each element corresponds to one arrow in denoted with the same letter   
The identity of • is the morphism associated with the unit element .  
Composition of arrows is given by multiplying the corresponding elements.  
Associativity and unitality of monoid law, translate directly to associativity and unit laws of arrows which make the delooping BM a category.

A monoid (or a group) can always be thought of as a category with a single object.

The category **Set** is the category whose objects are sets, and whose morphisms are maps (functions) between them.

A **functor** consists of:

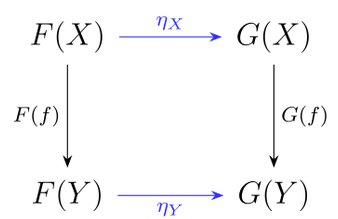
Intuition: A functor between two categories, is a function mapping objects between the two categories, together with a function mapping arrows between the two categories, where an arrow is sent to an arrow with the correctly mapped source and target objects, such that:  
composition is respected (composing then mapping, is the same as mapping then composing) and identities are respected (mapped to identities).  
A functor between two categories is a picture of the source category into the target category.

A **presheaf on a category C** is a functor from the opposite category to the Set category.  
It’s a function mapping objects of C to sets, together with a function mapping arrows to functions, where an arrow is sent to a function with correctly mapped source and target (in inverted order), such that composition is respected and identities are respected.

The **category Top** is the category where objects are topological spaces, and arrows are continuous functions between them.  
Given a fixed topological space , the set of open sets on with inclusion is a preordered set which gives a category also written .

The **standard presheaf example** is a presheaf on e.g. a functor from where:  
Each open set is mapped to the set: of continuous real valued functions defined on the open set.   
Each arrow (inclusion) is mapped to the function: which consists in restricting to .  
Here represents the ways occurs inside the presheaf.

**Top•** Is the category of pointed topological spaces where:  
An object is a couple where is a topological space and is some point of that space.  
An arrow is a continuous function between two pointed topological spaces that maps the first point on the second point.  
A functor Top• Grp can be obtained by sending a pointed topological space to the corresponding fundamental group and by sending a continuous function between pointed topological spaces, to the group morphism that consists in sending a loop from the first space to the other by applying the function on it.

A **natural transformation** is something defined between two functors , over the same two categories and . It consists of a function called the component, which sends a given an object from the source category, to an arrow in the target category between the images of the object by the two functors: . The component must satisfy the property that for every arrow in the source category, , that is :  
If we map that arrow with the first functor and apply afterwards the component at the arrow’s destination, we get the same as if we map the arrow with the second functor and apply before the component at the arrow’s source.  
The component is thus a system of target arrows associated with source objects, which give us a way to get an arrow from an arrow in a “natural” way.  
A natural transformation is a consistent system of arrows between the images of two functors.

**Cat** is the category whose objects are small categories, and arrows are the functors between them.

Given two categories then called the **functor category**, is the category where objects are to functors and arrows are natural transformations between them. It is a category.

A functor is **faithful** if it is “injective” meaning   
A functor is **full** if it is “surjective” meaning for every arrow in the target category there exists some source arrow such that .

A natural transformation between two functors , is a **natural isomorphism** if the component at every object (the target arrow) is an isomorphism.

A **covariant** functor is just a regular functor.  
A **contravariant** functor is a covariant functor. (which is the same as )

The **covariant hom-functor** in a category at a fixed object denoted is the functor that maps each object of to the corresponding set (of all arrows). (It maps each object to the collection of arrows into it coming from the fixed object)  
And it maps each arrow to a function (a Set arrow) .  
(It maps a given arrow to the function that maps arrows into ’s source to an arrow into ’s dest by composing it with the given arrow )  
The covariant hom-functor at is “all arrows from ”

The **contravariant hom-functor** on a category at a fixed object denoted is the presheaf on that maps each object to the collection of arrows from it going to the fixed object, and that maps each given arrow to the function that maps an arrow leaving ’s target to an arrow leaving ’s source by precomposing it with the given arrow. ( )  
The contravariant hom-functor at is “all arrows into ”

A functor from a category C to the Set category is **representable** iff it is naturally isomorphic to a covariant hom-functor of the category C at some fixed object. The pair consisting of the fixed object and the natural isomorphism is called a representation of the functor.

A presheaf being a contravariant functor from a category to Set, is representable iff it’s naturally isomorphic to a contravariant hom-functor at some fixed object.

Yoneda lemma in its contravariant version essentially says that:   
Given a category , and a presheaf on that category, and a fixed object .  
The natural transformations between the presheaf and arrows into (contra. hom functor), are in one-to-one correspondence with the image of by the presheaf.

For example, a natural transformation between the standard presheaf on , and open subsets of an open set , corresponds to a continuous function .

If for we choose the contravariant hom-functor at “all arrows into ”, we get:  
The natural transformations between “arrows into ” and “arrows into ” are in 1-to-1 correspondence with the “arrows from to ”. In other words, an arrow from to corresponds to a natural transformation from “arrows into ” to “arrows into ”.  
As a result, an isomorphism between and correspond to a natural isomorphism between arrows into and arrows into .   
In other words and have “the same structure” iff the arrows into A and arrows into have “the same structure”.

**Bifunctors**

The **product of two categories** has pairs as objects, and arrow pairs as arrows.  
A bifunctor is a functor from a product of two categories.  
Given a bifunctor, swapping its arguments yields another bifunctor.  
Given a bifunctor, fixing either argument yields a functor.  
A bifunctor can be curried into a functor   
A functor can be uncurried into a bifunctor   
The **evaluation bifunctor** maps an object and a functor, to applying the functor to the object.  
For example it can be seen as   
The **yoneda bifunctor** also written can be understood as   
Mapping two functors of to the set of natural transformations between them defines a bifunctor   
The **pairing bifunctor** is X, F ↦ (hom(\_,X) ⟹ F)