

:ROAM<sub>REFS</sub>: [https://docs.google.com/presentation/d/1rpqDXysh8GJJodXCTbWCj4MmIx87mT\\_ksyMqHoq-hkk/edit](https://docs.google.com/presentation/d/1rpqDXysh8GJJodXCTbWCj4MmIx87mT_ksyMqHoq-hkk/edit)

## 1 | Logicomix

## 2 | Category Theory

- Very general formalism of *objects* and *morphisms* - things and ways to connect them.
- Almost like a 'theory of everything' for math.

### 2.1 | Origins

- At some point Feynman realized that it's useful to draw linear operators as diagrams in a systematic manner.
- Later, Penrose realized that this can be applied more broadly. Representing linear operators in a topological/geometric manner became its own field.
- Separately, logicians had started using categories where the objects were propositions and morphism were proofs.
  - Programmers started using them where objects are data types and morphism are programs.
- Category theory describes all these things in an abstract manner
  - Focus on approaching these relationships and how to compose them in a way abstracted away from objects
- Curry-Howard correspondence
  - Person that Haskell is named after
  - Modeling programs as proofs

### 2.2 | Rosetta Stone

Category Theory	Physics	Topology	Logic	Computation
object	system	manifold	proposition	data type
morphism	process	cobordism	proof	program

### 2.3 | Intro to Categories

A *category*  $C$  consists of:

- a collection of *objects*, where if  $X$  is an object  $X \in C$ , and
- for every pair of objects  $(X, Y)$ , there is a set  $\text{hom}(X, Y)$  of morphisms from  $X$  to  $Y$ . We call this set  $\text{hom}(X, Y)$  a **homset**. If  $f \in \text{hom}(X, Y)$  then  $f$  maps  $X$  to  $Y$ , or more formally,  $f : X \rightarrow Y$

such that

- for every object  $X$  there is an identity morphism that maps  $X$  to  $X$ .
- morphisms are composable: given  $f : X \rightarrow Y$  and  $g : Y \rightarrow Z$  there is a morphism  $gf : X \rightarrow Z$
- an identity morphism is both a left and a right unit for composition (identities are commutative)
- composition is associative

These objects and morphisms could be anything (see ??). Consider the symbols  $X$  and  $Y$  as sets capable of internal structure.

One example would be Haskell code. Take functions like `words` as morphisms and data types like strings as objects (objects can be different types!). Our output is another object.

```
words "Hello World"
-- ["Hello", "World"]

length ["I", "like", "cats"]
-- 3
```

Just like a normal morphism, we can *compose* functions in Haskell.

```
let countwords = length . words
countwords "Yay for composition!"
-- 3
```

A more casual definition of a category is a *network of composable relationships*... of anything. The generality of category theory allows it to be useful to a wide variety of fields.

## 2.4 | Functors

A *functor*  $F : C \rightarrow D$  from a category  $C$  to a category  $D$  is a map sending:

- any object  $X \in C$  to an object  $F(X) \in D$ ,
- any morphism  $f : X \rightarrow Y$  in  $C$  to a morphism  $F(f) : F(X) \rightarrow F(Y)$  in  $D$

in such a way that:

- $F$  preserves identities
- $F$  preserves composition

The primary difference here is how functors can act on both objects and morphisms.

In logic, a category is a *theory* and the functor a *model* of a theory (allowing you to create a representation in another space).

## 2.5 | Natural Transformation

Gives us a way to turn functors into one another.

Given two functors  $F, F' : C \rightarrow D$  a natural transformation  $\alpha : F \Rightarrow F'$  assigns to every object  $X$  in  $C$  a morphism  $\alpha_x : F(x) \rightarrow F'(X)$  such that for any morphism  $f : X \rightarrow Y$  in  $C$ , the equation  $\alpha_y F(f) = F'(f) \alpha_x$  holds in  $D$ .

## 2.6 | Types of Categories

### 2.6.1 | Monoidal Categories

**review**

A monoidal category has a function  $\otimes : C \times C \rightarrow C$  (a product of categories outputs pairs of morphisms and pairs of objects??) that takes two objects and puts them together to give a new object  $X \otimes Y$ . This allows execution of morphisms in *parallel* or *series* by acting on pairs.

### 2.6.2 | Braided Monoidal Categories

**unresearched**

### 2.6.3 | Closed Categories

**unresearched**

$\text{hom}(X \otimes Y, Z) \cong \text{hom}(Y, X \rightarrow Z)$  Idea of turning an operation into an object? Connection to currying.

## 2.7 | Lambda Calculus

**unresearched**