Categorical Modeling Language

We define the simplest possible concept as that which distinguishes *some* **thing** from *no* **thing** – an object alone in the void:

$$\bullet$$

This abstraction captures the concept of *existence* itself. It is a visual depiction of the assertion: **I am**.

We need a way to refer to this object and distinguish it from other objects. **Identity** takes the form of a self-directed, labelled **arrow**.

$$A \to \bullet$$

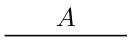
The `identity` arrow (`id` for short) acts like a pointer that states *this thing right here has the identity* `A`.

But an arrow is not a connection between an identity and an object. Arrows only connect objects, so the `identity` of `A` is really a labelled *line* `A` between an object and itself (i.e., a loop). This is denoted as:

$$\mathrm{id}_A : A \to A$$

```
id(A)
```

Since an object is known only by its identity arrow, we can simply use the arrow itself to represent the object `A`:




```
to_tikz(id(A), labels=true)
```

Discrete Categories

An object, depicted by its identity arrow, forms our first example of a **category**. A category acts like a *bounded context* for us to consider a collection of objects and, more importantly, a collection of arrows.

This particular category is called *discrete* and, since it has only one object (arrow), it is denoted: **Disc(1)**. This category captures the essence of what it means to be an *individual*. It isolates the concept, giving it a context, and discards all other considerations.

There could of course be more than one object:

$$\underline{\quad A \quad}$$

$$\underline{\quad B \quad}$$

```
to_tikz(id(A)⊗id(B), labels=true)
```

This forms another category, **Disc(2)**, which captures the essence of what it means to be a **Pair** – two *things* (`A`,`B`) considered together as *one* (the category `Disc(2)`). As you might imagine, there are many such discrete categories representing arbitrary collections of distinct *things* (i.e., *n*-tuples).

The essential ingredients of category theory are: **objects**, **arrows**, and **categories**, along with a few operations for combining arrows and a few rules that must be obeyed. (A more formal description is given below.)

Patterns of Meaning

Our primary focus is on the **patterns** of objects, arrows, and contexts (categories), each of which has a name (often daunting and cryptic) and its own set of rules, conditions, and implications. We've already seen the patterns (concepts) of *one* (an individual) and *two* (a pair). Concepts are captured and encoded in this language of patterns. They identify repeating conceptual arrangements in different contexts and help us to reason about situations and gain new insights. It enables us to build a precise description with well-understood and formally verified semantics (meaning; iterpretation).

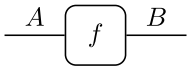
Relatedness

So far we've only discussed *discrete* categories (i.e., those with only objects and identity arrows). We suggested earlier that arrows connect objects. We've only seen `identity` arrows that connect the same object, but if we take `Disc(2)` and add an arrow (making the new category *non*-discrete) between `A` and `B`, let's call it `f`, we can capture the concept of an **Arrow** itself, as denoted:

$$f : A \to B$$

```
f
```

and depicted as:




```
to_tikz(f, labels=true)
```

This pattern captures the essence of a *from/to* relationship between two objects (`A`,`B`). It indicates that `A` is *related* to `B` in some way (`f`). The objects could be represent anything: *From* and *To*, *Source* and *Target*, or two other concepts, *Remus* and *Romulus*, while the arrow might indicate mathematical relations (e.g., $<$, $\times 2$), spatial relations (e.g., *atop*, *within*), familial relations (e.g., *hasBrother*, *hasAunt*), etc.

Composition

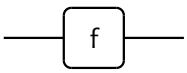
One of the important rules of arrows within a category is that they must compose together (i.e., form chains of to/from).

```
md"
### Composition

One of the important rules of arrows within a category is that they must compose
together (i.e., form chains of to/from).
"
```

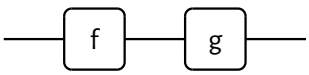
Symmetric monoidal category

$$(f : A \to B, \quad g : B \to A)$$

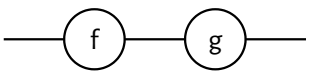
```
begin
    A, B, C, D = Ob(FreeSymmetricMonoidalCategory, :A, :B, :C, :D)
    f, g = Hom(:f, A, B), Hom(:g, B, A);
end

```



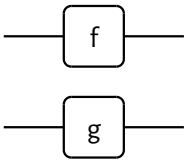

```
to_composejl(f)
```




```
to_composejl(f⋅g)
```




```
to_composejl(f⋅g, default_box_shape=:circle)
```



- `to_composejl(f⊗g)`