1 Definition

A category ζ is a thing that is made up of:

- **Objects**

  A set of objects $Ob(\zeta)$

  Notes

  - These objects could be at any level of abstraction and for the definions and working of category theory, we don't need to look under that abstraction. I think this is why category theory becomes really powerful.

- **Morphisms**

  For every to elements c,d in $Ob(\zeta)$, there is a set of morphisms. This set is denoted by $\zeta(c, d)$ (or $Hom_\zeta(c, d)$) and contains morphisms $f \in \zeta(c, d)$, also denoted as $f : c \to d$

  The set of morphisms is also called *Hom-set* due to Category theory's origins in Homomorphisms

  Notes

  - In context or programming, it comes more naturally to think of morphisms as functions. But that is a special case which only applies on a special category (Types). To be able to exploit the power of abstractions in category theory, its important to remember that it could be anything. For example, in a relational DB, it could be a foreign key.
  - I was initially confused whether its necessary to always have a Hom-set for any two objects in a category, but then I realized that just because there is a Hom-set, it doesn't mean it has any members - it could be a null set too. So that way, of course, there is always a Hom-set.

- **Identities**

  Each object c in a category must have an identity morphism, which would be an element of $\zeta(c, c)$ for all $c \in Ob(\zeta)$. An identity morphism is a morphism that satisfies the *unitality* constraint (defined below)

  Notes

  - What if there are multiple morphisms that satisfy the unitality constraint? Is identity unique? If so, how to select one of the candidates?

- **Composition**

  The morphisms in the category must be composable. For any three objects c, d, e in the category, and corresponding morphisms $f : c \to d$ and $g : d \to e$, there always exists a morphism $h : c \to e$ which is a composition of f and g

  $h := g \circ f$

  The compositions must satisfy the *Associativity* constraint (defined below)

  Notes

  - In context of programming language implementation, is it necessary for the runtime to always define a new function for every occurrence of composition? What are the challenges there (both if its necessary, or not)?

2 Constraints

- **Unitality**

  For any $f : c \to d$, the identities id_c and id_d which are identities for objects c and d respectively,

  $f \circ id_c = f = id_d \circ f$

- **Associativity**

  For any $f : a \to b$, $g : b \to c$ and $h : c \to d$,

  $h \circ (g \circ f) = (h \circ g) \circ f$

  This constraint would make the usage of paranthesis in composition unnecessary.

  Notes

  - This might sound unnecessary to specify the associativity because many examples come to mind which are associative so it might start feeling like everything is associative, but its quite an important constraint. For example, it prohibits exponents to be considered morphism because $(a^b)^c$ is not always same as $a^{(b^c)}$, so they dont compose without order being important.