Both the type and sole value of the empty tuple.

Types start with an uppercase character, variables with a lowercase letter.

Type variables stay maximally general (polymorphic), being restricted only by the ways the type is used. Generics force you to anticipate the maximum generality yourself when declaring the generic type's bounds.

... an interface. It creates a contract that defines the behaviors of types of that class.

... its name consists of only special characters and is infix by default.

It indicates a class constraint. The left hand side indicates class membership/s (comma separated) of the type variables used on the right hand side.

The type class for types that can be tested for equality.

`IO` and functions are not members.

... `Ord`.

Its members can be presented as strings using the `show` function.

The `Read` typeclass defines `read` which does just that.

However, in order to know what type is desired, the result of `read` must be used in an expression.

Haskell's term for a manifest type. It's not an annotation on a type in the sense of Scala's `@specialized`.
Haskell type annotations use the `::` followed by the type.

They can be enumerated.

They have an upper and a lower bound: `maxBound` and `minBound`.

```
ghci> :t 0
0 :: (Num t) => t
ghci> :t maxBound
maxBound :: (Bounded a) => a
```

```
ghci> :t fromIntegral
fromIntegral :: (Integral a, Num b) => a -> b
```