> *It's not pining. It's passed on. This parrot is no more. It has ceased to be. It's expired and gone to meet its maker. This is a late parrot. It's a stiff. Bereft of life, it rests in peace. If you hadn't nailed it to the perch, it would be pushing up the daisies. It's rung down the curtain and joined the choir invisible.*
> *THIS IS AN EX-PARROT.*

Abstract Data Types

So far, we have looked at processing numbers. For most non-trivial problems, we need more sophisticated data structures than numbers. In this section we look at the idea of **abstract data types (ADTs)** and we will be using strings as an example of an ADT as we have already been dealing with them.

An abstract data type is a collection of data and operations on the data. The behaviour of the ADT and its operations is specified independently of any particular implementation. This is separation of concerns is an important concept in software engineering – as it completely separates how the ADT is used from its implementation.

The Abstraction

An implementor (designer) of the ADT provides an **interface** to the data that satisfies the specification of the ADT. The user of the ADT relies on the interface for their own programs without having to worry about how it is implemented. This has two very important benefits – firstly, the user does not have to worry about the details of the implementation, and secondly, the implementor is free to change the implementation (to, for example, make it more efficient) without having any effect on the correctness of user code. This is useful as the user of the ADT can write code using the ADT and the implementor can change the implementation of the ADT and **the user's code will still work**. It is called abstract because the user has an abstract, rather than concrete (implementation dependent), view of the data.

In a properly defined ADT, the supplied interface is the *only* way to **cross the abstraction barrier**. In other words, the user of the ADT should not be able to access or modify data directly. If a user was able to do this then a change of implementation can invalidate user code.

An ADT typically consists of:

- **constructors** – these build instances of the ADT from user supplied values
- **accessors** – these access components of the ADT
- **mutators** – these modify data stored in the ADT
- **tests** – these test if the ADT has particular properties

In an object-oriented language like Python, everything is an object (as we will see later). Designing and using ADTs is really a case of designing and using classes. In the object-oriented world, an ADT is just a specific kind of class. Strings in Python are objects written in a way that they can be thought of and used as an ADT.