Blocks also give us a basic error-handling functionality, the idea being that we specify blocks to be executed when something goes wrong. For instance, in a `Collection` object (a container of objects) the method `remove:` will try to remove the specified element from the collection. The method `remove:ifAbsent:` will try to remove the specified element from the collection, and if the element does not exist, it will execute the block passed as argument in `ifAbsent:`. In a nice example of minimizing code, the first is implemented in terms of the second:

```
remove: oldObject
    "Remove oldObject from the receiver's elements. Answer oldObject
    unless no element is equal to oldObject, in which case, raise an error.
    ArrayedCollections cannot respond to this message."

    ^ self remove: oldObject ifAbsent: [self errorNotFound: oldObject]

remove: oldObject ifAbsent: anExceptionBlock
    "Remove oldObject from the receiver's elements. If several of the
    elements are equal to oldObject, only one is removed. If no element is
    equal to oldObject, answer the result of evaluating anExceptionBlock.
    Otherwise, answer the argument, oldObject. ArrayedCollections cannot
    respond to this message."

    self subclassResponsibility
```

`self` is a reference to the current object (equivalent to `this` in C++ and Java); it is a reserved name, a *pseudovariable* with fixed semantics. There are some more pseudovariables: `super` is a reference to the superclass (equivalent to `super` in Java); `nil`, `true`, and `false` have the expected meanings; and finally there is also `thisContext`, which we see in action in the definition of the `subclassResponsibility` method. This selector is defined in `Object` and is simply an indicator that the subclass must override it:

```
subclassResponsibility
    "This message sets up a framework for the behavior of the class's subclasses.
    Announce that the subclass should have implemented this message."

    self error: 'My subclass should have overridden ', thisContext
 sender selector printString
```

The `thisContext` pseudovariable is a reference to the current executing context—that is, the current executing method or block—so it is an instance of the current executing instance of `MethodContext` or `BlockContext`. The `sender` selector returns the context that sent the message; `selector` gives the selector of the method. *All these are objects.*

Handling code as objects is not new; a strength of Lisp is its similar handling of code and data. It allows us to program using reflection—that is, to do metaprogramming.

Metaprogramming is an idea whose importance has been increasing with time. In statically typed compiled languages such as C or C++, support for metaprogramming is meager. An effort to put metaprogramming to work in C++ (Forman and Danforth 1999), seems to have been more influential in Python than in C++ itself (see PEP 253, "Subtyping Built-in Types," at