8.12. Defining an Interface or Abstract Base Class [Book]

Problem

You want to define a class that serves as an interface or abstract base class from which you can perform type checking and ensure that certain methods are implemented in subclasses.

Solution

To define an abstract base class, use the `abc` module. For example:

```
from abc import ABCMeta, abstractmethod

class IStream(metaclass=ABCMeta):
    @abstractmethod
    def read(self, maxbytes=-1):
        pass
    @abstractmethod
    def write(self, data):
        pass
```

A central feature of an abstract base class is that it cannot be instantiated directly. For example, if you try to do it, you'll get an error:

```
a = IStream()   # TypeError: Can't instantiate abstract class
                # IStream with abstract methods read, write
```

Instead, an abstract base class is meant to be used as a base class for other classes that are expected to implement the required methods. For example:

```
class SocketStream(IStream):
    def read(self, maxbytes=-1):
        ...
    def write(self, data):
        ...
```

A major use of abstract base classes is in code that wants to enforce an expected programming interface. For example, one way to view the `IStream` base class is as a high-level specification for an interface that allows reading and writing of data. Code that explicitly checks for this interface could be written as follows:

```
def serialize(obj, stream):
```

```
    if not isinstance(stream, IStream):
        raise TypeError('Expected an IStream')
    ...
```

You might think that this kind of type checking only works by subclassing the abstract base class (ABC), but ABCs ...