

Python Coding Idioms pt 1 – class interfaces

Tennessee Leeuwenburg

The first things to come to grips with in any programming language are its basics – how to achieve tasks like calling a method, creating a class, reacting to user input and creating data structures.

This can get you so far, but without some more subtle tools in the box, it can leave you with a series of ad-hoc solutions which do not read naturally (by this I mean have obvious meaning) and, quite likely, result in inefficient behavior.

This article will concentrate on *coding idiom*, in particular that of using consistent ways for declaring methods within python code for indicating additional information about their intended use. By adopting this idiom it is possible to tell at a glance which methods of a class are intended for public consumption, and which relate more to internal use.

Let's jump straight into an example (see Text 2).

```

1: class NewPayment:
2:
3: def __init__(self, payment_info, callbacks):
4:     self.payment_info = payment_info
5:     self.callbacks = callbacks
6:
7: def process(self):
8:     """
9:     Processes the payment into the financial system
10:    """
11:
12:     self.recordPayment()
13:     self.despatchOrder()
14:     self.sendConfirmation()
15:     self.concludePayment()
16:
17: def recordPayment():
18:     """
19:     Make a permanent record of the payment in the database
20:    """
21:     pass
22:
23: def despatchOrder():
24:     """
25:     Despatch order to the processing subsystem
26:    """
27:     pass
28:
29: def concludePayment():
30:     """
31:     Clean up object references, call any callbacks
32:    """
33:
34:     for callback in self.callbacks:
35:         callback.callback(self)

```

Text 2: Initial ordering system without suggested notation

underscore

This is consistent with PEP-8¹, and is basically a subset of it. PEP-8's goals are to cover coding style, which is a larger topic than this article can tackle.

To make a brief diversion, however, coding style is something which is often only appreciated after some time spent programming. This is because its effects are seldom seen within small or self-contained projects, but rather become apparent when examining code years later, adapting someone else's code, or in small efficiency gains which come from having good programming habits generally.

id • i • om (n.)

2.2. The specific grammatical, syntactic, and structural character of a given language.

...

5. A style of artistic expression characteristic of a particular individual, school, period or medium: the idiom of the French impressionists; the punk rock idiom.

Text 1: Source: dictionary.com

Here we can see a class which represents a payment in a fictional ordering system. The payment class contains a trigger for its processing and rules for performing that processing.

What is not obvious at first glance is which of those methods are intended for other aspects of the system to interact with directly, and which of those methods are related to the internal processing logic. This article will describe a habit which, if adopted, can provide an at-a-glance way to make that distinction. Such a distinction is useful in learning how to use a new piece of code (for example, extending and inheriting from an unfamiliar class) and also for documentation purposes. The effects of this on the final code will also be shown.

In short:

- 'Interface' methods are presented without a prefix
- 'Internal' methods which may be over-ridden are prefixed by a single underscore
- 'Private' methods are prefixed by a double underscore

What's a PEP?

A PEP is a Python Enhancement Proposal. These documents make up what information is available to the community, covering new features, design decisions and other aspects of Python development. They are available at <http://www.python.org/dev/peps/>

1 "Style Guide for Python Code", Guido van Rossum: <http://www.python.org/dev/peps/pep-0008/>

