Функциональные интерфейсы и код, основанный на обратных вызовах

В качестве примера отметим, что комплект инструментов для построения графических пользовательских инструментов `tkinter` (`Tkinter` в Python 2.X) позволяет регистрировать функции как обработчики событий (так называемые *обратные вызовы*) — когда возникают события, `tkinter` вызывает зарегистрированные объекты. Если вы хотите, чтобы обработчик событий сохранял состояние между событиями, тогда можете зарегистрировать либо *связанный метод* класса, либо экземпляр, который с помощью `__call__` обеспечивает соответствие ожидаемому интерфейсу.

Скажем, в коде предыдущего раздела `x.comp` из второго примера и `x` из первого могут таким способом передавать объекты, подобные функциям. *Функции замыканий* с состоянием из объемлющих областей видимости, рассматриваемые в главе 17 первого тома, способны достигать похожего эффекта, но не обеспечивают столько поддержки для множества операций или настройки.

В следующей главе связанные методы обсуждаются более подробно, а пока ниже представлен гипотетический пример метода `__call__` применительно к области графических пользовательских интерфейсов. Приведенный далее класс определяет объект, поддерживающий интерфейс вызова функций, но также запоминающий цвет, который должна получить кнопка при щелчке на ней в будущем:

```
class Callback:
    def __init__(self, color):              # Функция + информация о состоянии
        self.color = color
    def __call__(self):                     # Поддерживает вызовы без аргументов
        print('turn', self.color)
```

В контексте графического пользовательского интерфейса мы можем зарегистрировать экземпляры этого класса в качестве обработчиков событий для кнопок, хотя графический пользовательский интерфейс рассчитывает на возможность вызова обработчиков событий как простых функций без аргументов:

```
# Обработчики
cb1 = Callback('blue')                      # Запомнить blue
cb2 = Callback('green')                     # Запомнить green

B1 = Button(command=cb1)                    # Зарегистрировать обработчики
B2 = Button(command=cb2)
```

Когда позже на кнопке совершается щелчок, объект экземпляра вызывается как простая функция без аргументов подобно показанным ниже вызовам. Тем не менее, поскольку этот объект сохраняет состояние в виде атрибутов экземпляра, он запоминает, что делать, становясь объектом *функции с состоянием*:

```
# События
cb1()                                       # Выводит turn blue
cb2()                                       # Выводит turn green
```

В действительности многие считают такие классы лучшим способом предохранения информации о состоянии в языке Python (во всяком случае, согласно общепринятым принципам стиля Python). С помощью ООП запоминание состояния делается явным посредством присваивания значений атрибутам. Данный способ отличается от других методик предохранения состояния (например, глобальные переменные, ссылки из объемлющих областей видимости и изменяемые аргументы со стандартными значениями), которые полагаются на более ограниченное или менее явное поведе-