```
>>> print(x)                                    # Выполняется __str__
[Value: 4]
>>> str(x), repr(x)
('[Value: 4]', '<__main__.addstr object at 0x00000000029738D0>')
```

По этой причине метод `__repr__` может оказаться лучше, если вы хотите иметь *единственное* отображение для всех контекстов. Тем не менее, за счет определения обоих методов вы можете поддерживать в разных контекстах отличающиеся отображения – например, отображение посредством `__str__` для конечного пользователя и низкоуровневое отображение с помощью `__repr__` для применения программистами во время разработки. В действительности `__str__` просто переопределяет `__repr__` для контекстов отображения, более дружественных к пользователю:

```
>>> class addboth(adder):
        def __str__(self):
            return '[Value: %s]' % self.data   # Строка, дружественная
                                               #   к пользователю
        def __repr__(self):
            return 'addboth(%s)' % self.data   # Строка как в коде

>>> x = addboth(4)
>>> x + 1
>>> x                                          # Выполняется __repr__
addboth(5)
>>> print(x)                                   # Выполняется __str__
[Value: 5]
>>> str(x), repr(x)
('[Value: 5]', 'addboth(5)')
```

Замечания по использованию отображения

Несмотря на простоту использования в целом, я должен привести три замечания относительно этих методов. Во-первых, имейте в виду, что `__str__` и `__repr__` обязаны возвращать *строки*; другие результирующие типы не преобразуются и вызывают ошибки, так что при необходимости обеспечьте их обработку инструментом преобразования в строку (скажем, `str` или `%`).

Во-вторых, в зависимости от логики преобразования в строки дружественное к пользователю отображение `__str__` может применяться, только когда объекты находятся на верхнем уровне операции `print`; объекты, *вложенные* внутрь более крупных объектов, могут по-прежнему выводиться посредством `__repr__` либо их стандартных методов. Оба аспекта иллюстрируются в следующем взаимодействии:

```
>>> class Printer:
        def __init__(self, val):
            self.val = val
        def __str__(self):                 # Используется для самого экземпляра
            return str(self.val)           # Преобразовать в строковый результат

>>> objs = [Printer(2), Printer(3)]
>>> for x in objs: print(x)                # __str__ выполняется при выводе экземпляра
                                           # Но не в ситуации, когда экземпляр находится в списке!
2
3
>>> print(objs)
[<__main__.Printer object at 0x000000000297AB38>, <__main__.Printer obj...
etc...>]
```