Из-за того, что ListInstance определяет метод перегрузки операции __str__, создаваемые из этого класса экземпляры автоматически отображают свои атрибуты при выводе, давая чуть больше информации, чем простой адрес. Ниже показана работа класса в режиме одиночного наследования, когда он подмешивается к классу из предыдущего раздела (код выполняется в Python 3.X и 2.X одинаково, хотя repr из Python 2.X по умолчанию отображает метку instance, а не object):

```
>>> from listinstance import ListInstance
>>> class Spam(ListInstance):          # Наследует метод __str__
        def __init__(self):
            self.data1 = 'food'

>>> x = Spam()
>>> print(x)                           # print() и str() запускают __str__
<Instance of Spam, address 43034496:
        data1=food
>
```

Вы можете также извлекать и сохранять выходной список в виде строки, не выводя его посредством str, а эхо-вывод интерактивной подсказки по-прежнему применяет стандартный формат, потому что возможность реализации метода __repr__ мы оставили клиентам как вариант:

```
>>> display = str(x) # Вывести строку для интерпретации управляющих символов
>>> display
'<Instance of Spam, address 43034496:\n\tdata1=food\n>'

>>> x                  # Метод __repr__ по-прежнему использует стандартный формат
<__main__.Spam object at 0x000000000290A780>
```

Класс ListInstance пригоден для любого создаваемого класса — даже классов, которые уже имеют один и более суперклассов. Именно здесь пригодится *множественное наследование*: за счет добавления ListInstance к перечню суперклассов в строках заголовка оператора class (т.е. подмешивая класс ListInstance) вы получаете его метод __str__ "бесплатно" наряду с тем, что наследуете из существующих суперклассов. В файле testmixin0.py приведен пробный тестовый сценарий:

```
# Файл testmixin0.py
from listinstance import ListInstance  # Получить класс инструмента для
вывода списка атрибутов

class Super:
    def __init__(self):                # Метод __init__ суперкласса
        self.data1 = 'spam'            # Создать атрибуты экземпляра
    def ham(self):
        pass

class Sub(Super, ListInstance):        # Подмешивание ham и __str__
    def __init__(self):                # Классы, выводящие списки атрибутов,
                                       #   имеют доступ к self
        Super.__init__(self)
        self.data2 = 'eggs'            # Дополнительные атрибуты экземпляра
        self.data3 = 42
    def spam(self):                    # Определить здесь еще один метод
        pass

if __name__ == '__main__':
    X = Sub()
    print(X)                           # Выполняется подмешанный метод __str__
```