Мы применяли ряд инструментов подобного рода в предыдущей главе, но чтобы подвести итог и помочь вам лучше понять, каким образом внутренне работают атрибуты, давайте запустим интерактивный сеанс, который отслеживает рост словарей пространств имен, когда задействованы классы. Теперь, обладая дополнительными знаниями о методах и суперклассах, мы можем предоставить здесь улучшенный обзор. Первым делом определим суперкласс и подкласс с методами, которые будут сохранять данные в своих экземплярах:

```
>>> class Super:
        def hello(self):
            self.data1 = 'spam'

>>> class Sub(Super):
        def hola(self):
            self.data2 = 'eggs'
```

Когда мы создаем экземпляр подкласса, он начинает свое существование с пустого словаря пространства имен, но имеет связи с классом для обеспечения работы поиска при наследовании. В действительности дерево наследования явно доступно в специальных атрибутах, которые можно исследовать. Экземпляры располагают атрибутом `__class__`, связывающим их с классом, а классы имеют атрибут `__bases__`, который представляет собой кортеж, содержащий связи с находящимися выше в дереве суперклассами (приведенный далее код запускался в Python 3.7; ваши форматы имен, внутренние атрибуты и порядок ключей могут отличаться):

```
>>> X = Sub()
>>> X.__dict__                  # Словарь пространства имен экземпляра
{}
>>> X.__class__                 # Класс экземпляра
<class '__main__.Sub'>
>>> Sub.__bases__               # Суперкласс класса
(<class '__main__.Super'>,)
>>> Super.__bases__             # Пустой кортеж () в Python 2.X
(<class 'object'>,)
```

Когда классы выполняют присваивание атрибутам `self`, они заполняют объекты экземпляров – т.е. атрибуты оказываются в словарях пространств имен экземпляров, а не классов. Пространство имен объекта экземпляра записывает данные, которые могут варьироваться от экземпляра к экземпляру, причем `self` является привязкой к этому пространству имен:

```
>>> Y = Sub()

>>> X.hello()
>>> X.__dict__
{'data1': 'spam'}

>>> X.hola()
>>> X.__dict__
{'data1': 'spam', 'data2': 'eggs'}

>>> list(Sub.__dict__.keys())
['__module__', 'hola', '__doc__']
>>> list(Super.__dict__.keys())
['__module__', 'hello', '__dict__', '__weakref__', '__doc__']

>>> Y.__dict__
{}
```