10.10. Objects and References

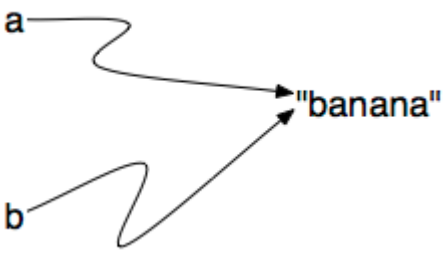
If we execute these assignment statements,

```
a = "banana"
b = "banana"
```

we know that `a` and `b` will refer to a string with the letters `"banana"` . But we don't know yet whether they point to the *same* string.

There are two possible ways the Python interpreter could arrange its internal states:

or



In one case, `a` and `b` refer to two different string objects that have the same value. In the second case, they refer to the same object. Remember that an object is something a variable can refer to.

We already know that objects can be identified using their unique identifier. We can also test whether two names refer to the same object using the *is* operator. The *is* operator will return true if the two references are to the same object. In other words, the references are the same. Try our example from above.

Run Show CodeLens

```
1 a = "banana"
2 b = "banana"
3
4 print(id(a) is id(b),"d")
5
```

letion.html) (ListDeletion.html) (Aliasing.html)

```
True d
```

ActiveCode: 1 (chp09_is1)

The answer is `True`. This tells us that both `a` and `b` refer to the same object, and that it is the second of the two reference diagrams that describes the relationship. Since strings are *immutable*, Python can optimize resources by making two names that refer to the same string literal value refer to the same object.

This is not the case with lists. Consider the following example. Here, `a` and `b` refer to two different lists, each of which happens to have the same element values.

2017/09/29, 11:52:23

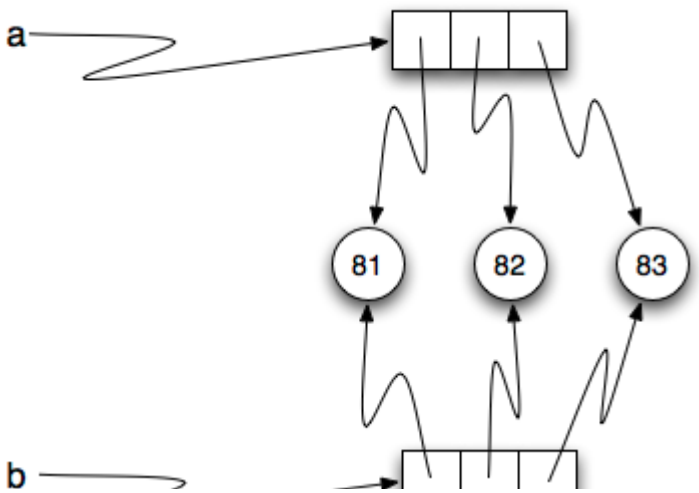
Run Show CodeLens

```
1 a = [81, 82, 83]
2 b = [81, 82, 83]
3
4 print(a is b)
5 print(id(a) is id(b))
6 print(a == b)
7
```

```
False
False
True
```

ActiveCode: 2 (chp09_is2)

The reference diagram for this example looks like this:



eletion.] (ListDeletion.) (Aliasing.)

`a` and `b` have the same value but do not refer to the same object.

There is one other important thing to notice about this reference diagram. The variable `a` is a reference to a **collection of references**. Those references actually refer to the integer values in the list. In other words, a list is a collection of references to objects. Interestingly, even though `a` and `b` are two different lists (two different collections of references), the integer object `81` is shared by both. Like strings, integers are also immutable so Python optimizes and lets everyone share the same object for some commonly used small integers.

Here is the example in codelens. Pay particular attention to the id values.

Python 2.7

```
1  a = [81, 82, 83]
2  b = [81, 82, 83]
3
4  print(a is b)
5  print(a == b)
```

<< First | < Back | Step 1 of 4 | Forward > | Last >>

line that has just executed
next line to execute

Frames Objects

CodeLens: 1 (chp09_istrace)

user not logged in

eletion.] (ListDeletion.) (Aliasing.)