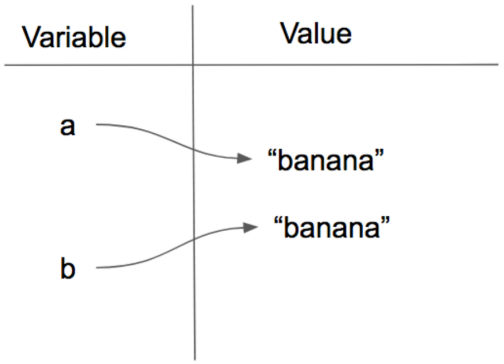
# 9.4. Objects and References

If we execute these assignment statements,

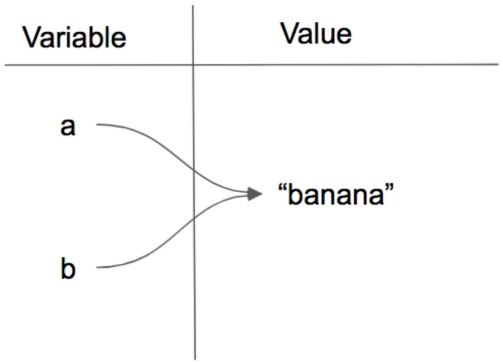
```
a = "banana"
b = "banana"
```

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 can 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.

Save & Run 5/13/2021, 1:55:35 PM - 2 of 2 Show in CodeLens

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

```
True
```

Activity: 1 -- ActiveCode (ac8_3_1)

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. Python assigns every object a unique id and when we ask `a is b` what python is really doing is checking to see if id(a) == id(b).

Save & Run 5/13/2021, 1:55:37 PM - 2 of 2 Show in CodeLens

```
1 a = "banana"
2 b = "banana"
3
```

```
print(id(a))
print(id(b))
```

```
2
2
```

Activity: 2 -- ActiveCode (ac8_3_2)

Since strings are *immutable*, the Python interpreter often optimizes resources by making two names that refer to the same string value refer to the same object. You shouldn't count on this (that is, use `==` to compare strings, not `is`), but don't be surprised if you find that two variables,each bound to the string "banana", have the same id..

This is not the case with lists, which never share an id just because they have the same contents. Consider the following example. Here, `a` and `b` refer to two different lists, each of which happens to have the same element values. They need to have different ids so that mutations of list `a` do not affect list `b`.

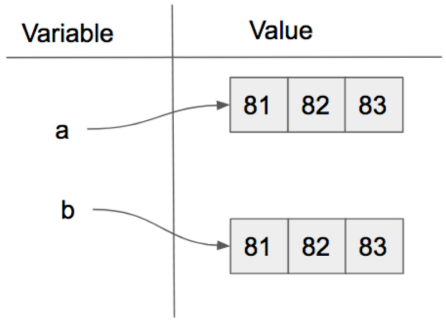
Save & Run 5/13/2021, 1:55:41 PM - 2 of 2 Show in CodeLens

```
a = [81,82,83]
b = [81,82,83]

print(a is b)

print(a == b)

print(id(a))
print(id(b))

```

```
False
True
3
4
```

Activity: 3 -- ActiveCode (ac8_3_3)

The reference diagram for this example looks like this:




`a` and `b` have equivalent values but do not refer to the same object. Because their contents are equivalent, a==b evaluates to True; because they are not the same object, a is b evaluates to False.

You have attempted 4 of 3 activities on this page

9.3. List Element Deletion">
Element Deletion">
9.5. Aliasing">

✔ Completed. Well Done!


| Back to top