PROGRAMMING, PYTHON

Optimization in Python — Interning

Understand Python's optimization technique — Interning

Chetan Ambi Aug 20, 2020 · 4 min read ★

Photo by George Stewart on Unsplash

There are different Python implementations out there such as *CPython*, *Jython*, *IronPython*, etc. The optimization techniques we are going to discuss in this article are related to *CPython* which is standard Python implementation.

Interning

Interning is re-using the objects on-demand instead of creating new objects. What does this mean? Let's try to understand Integer and String interning with examples.

> ***is*** *— this is used to compare the memory location of two python objects.*
> ***id*** *— this returns memory location in base-10.*

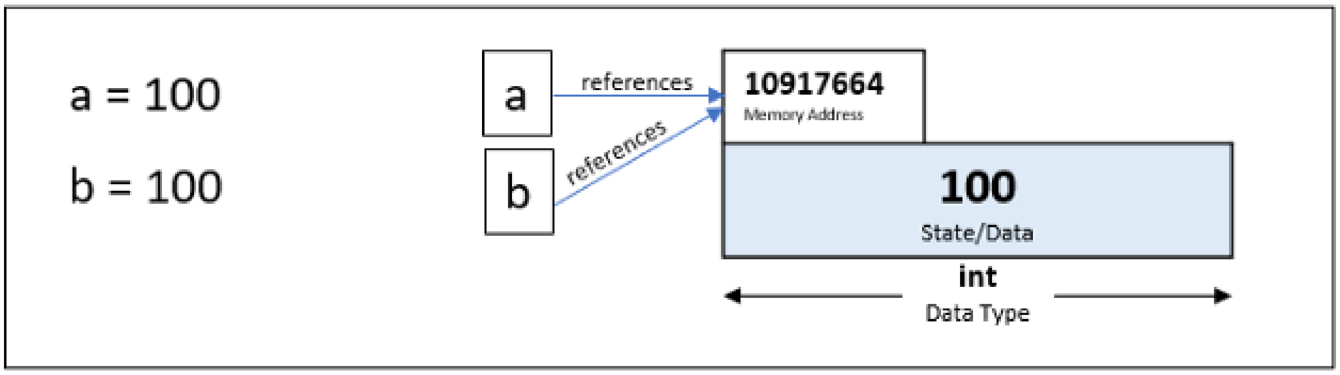
Integer interning

At startup, Python pre-loads/caches a list of integers into the memory. These are in the range `-5 to +256`. Any time when we try to create an integer object within this range, Python automatically refer to these objects in the memory instead of creating new integer objects.

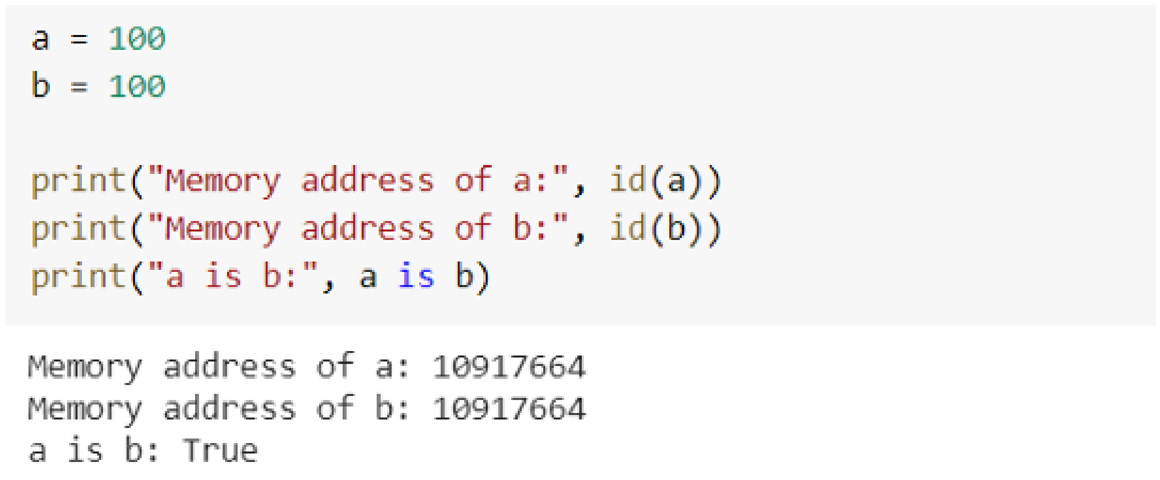
The reason behind this optimization strategy is simple that integers in the `-5 to 256` are used more often. So it makes sense to store them in the main memory. So, Python pre-loads them in the memory at the startup so that speed and memory are optimized.

Example 1:

In this example, both `a` and `b` are assigned to value 100. Since it is within the range `-5 to +256`, Python uses interning so that `b` will also reference the same memory location instead of creating another integer object with the value 100.



As we can see from the code below, both `a` and `b` are referencing the same object in the memory. Python will not create a new object but instead references to `a` 's memory location. This is all due to integer interning.



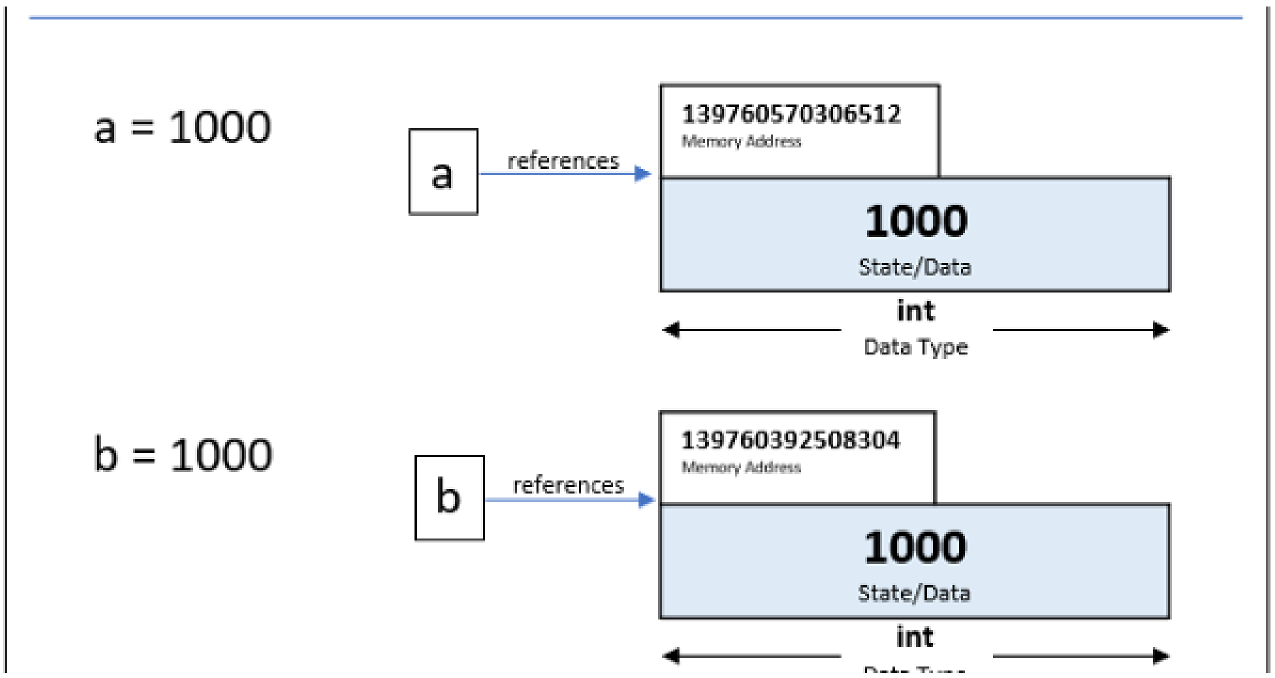
```
a = 100
b = 100

print("Memory address of a:", id(a))
print("Memory address of b:", id(b))
print("a is b:", a is b)
```

```
Memory address of a: 10917664
Memory address of b: 10917664
a is b: True
```

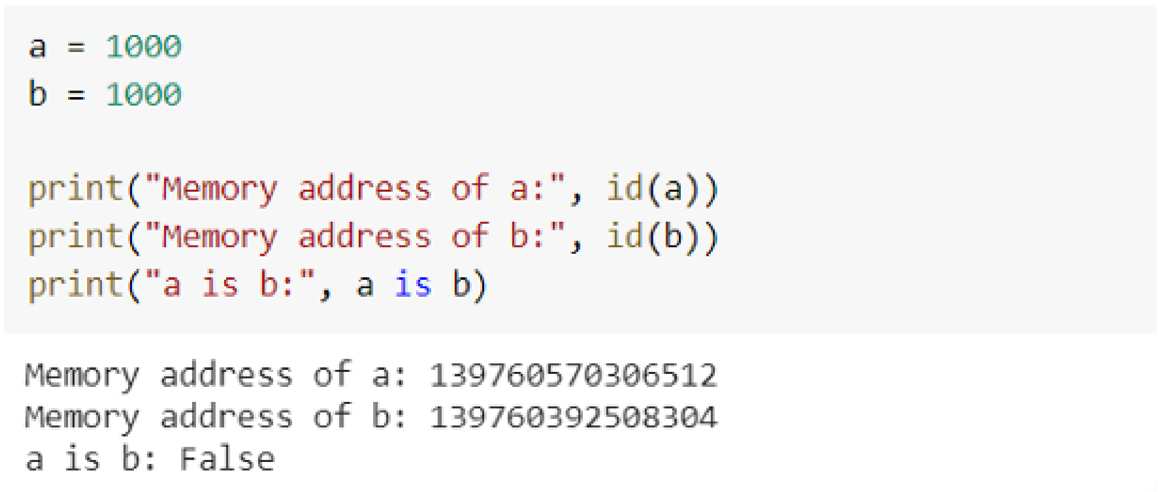
Image by Author

Example 2:

In this example, both `a` and `b` are assigned with value 1000. Since it is outside the range -5 to +256, Python will create two integer objects. So both a and b will be stored in different locations in the memory.



As we can see from the code below, both `a` and `b` are stored in different locations in the memory.



```
a = 1000
b = 1000

print("Memory address of a:", id(a))
print("Memory address of b:", id(b))
print("a is b:", a is b)
```

```
Memory address of a: 139760570306512
Memory address of b: 139760392508304
a is b: False
```

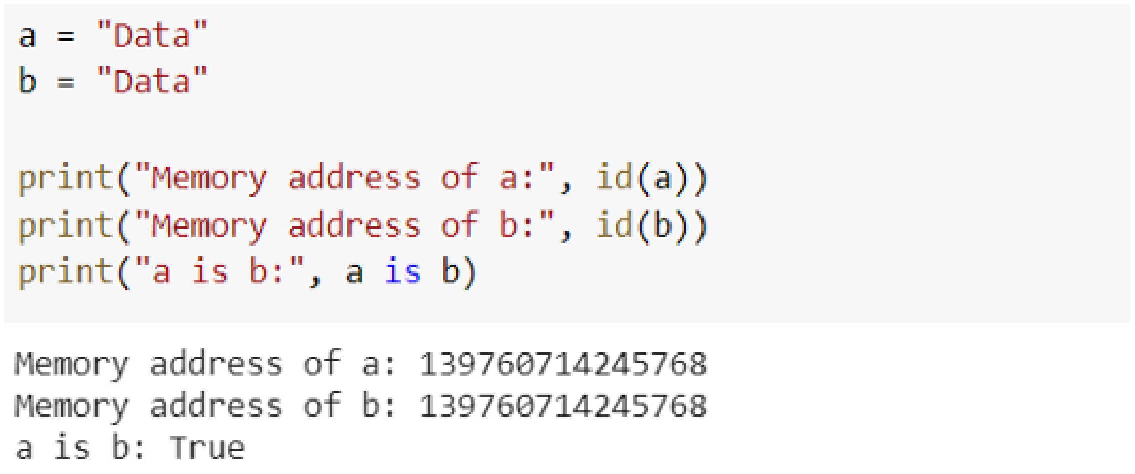
Image by Author

String interning

Like integers, some of the strings also get interned. Generally, any string that satisfies the identifier naming convention will get interned. Sometimes there will be exceptions. So, don't rely on it.

Example 1:

The string "Data" is a valid identifier, Python interns the string so both the variables will point to the same memory locations.

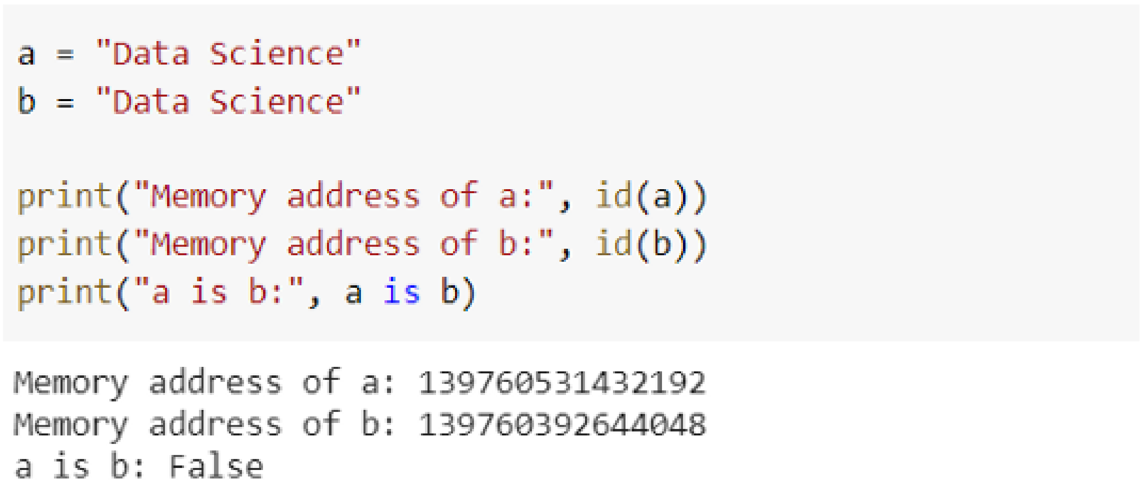


```
a = "Data"
b = "Data"

print("Memory address of a:", id(a))
print("Memory address of b:", id(b))
print("a is b:", a is b)
```

```
Memory address of a: 139760714245768
Memory address of b: 139760714245768
a is b: True
```

***Example 2*:**

The string “Data Science” is not a valid identifier. Hence string interning is not applied here so both a and b will point to two different memory locations.



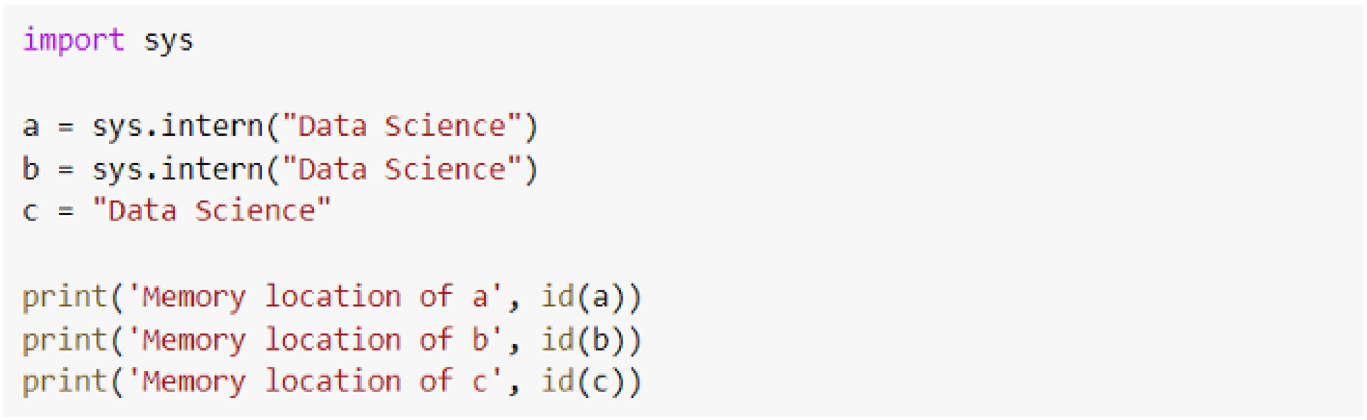
```
a = "Data Science"
b = "Data Science"

print("Memory address of a:", id(a))
print("Memory address of b:", id(b))
print("a is b:", a is b)
```

```
Memory address of a: 139760531432192
Memory address of b: 139760392644048
a is b: False
```

Image by Author

> *All the above examples are from Google Colab which has Python version 3.6.9*
>
> *In Python 3.6, any valid string with length ≤ 20 will get interned. But in Python 3.7, this has been changed to 4096. So as I mentioned earlier, these things will keep changing for different Python versions.*

Since not all strings are interned, Python provides the option force the string to be interned using `sys.intern()` . This should not be used unless there is a need. Refer the sample code below.



```
import sys

a = sys.intern("Data Science")
b = sys.intern("Data Science")
c = "Data Science"

print('Memory location of a', id(a))
print('Memory location of b', id(b))
print('Memory location of c', id(c))
```

Get started Open in app

Image by Author

Why string interning is important?

Let's assume that you have an application where a lot of string operations are happening. If we were to use `equality operator ==` for comparing long strings Python tries to compare it character by character and obviously it will take some time. But if these long strings can be interned then we know that they point to the same memory location. In such a case we can use `is` keyword for comparing memory locations as it works much faster.

Conclusion

Hope that you have understood the concept of *interning* for optimization in Python.

If you are interested in learning more about optimization go through the **Peephole optimization**.

Optimization in Python — Peephole

A brief introduction to Python's Peephole optimization technique

towardsdatascience.com

To understand M*utability and Immutability in Python*, please **click here** to read the article.

To read more such interesting articles on Python and Data Science, ***subscribe*** *to my blog* ***www.pythonsimplified.com*****.** You can also reach me on **LinkedIn**.

Thanks to Elliot Gunn.

Sign up for The Variable

By Towards Data Science

Python Programming