

# PYTHON OPTIMIZATIONS

INTERNING



## Important Note:

A lot of what we discuss with memory management, garbage collection and optimizations, is usually specific to the Python implementation you use.

In this course, we are using **CPython**, the standard (or reference) Python implementation (written in C).

But there are other Python implementations out there. These include:

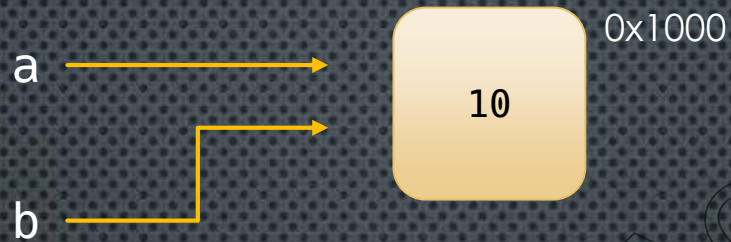
- **Jython** – written in Java and can import and use any Java class – in fact it even compiles to Java bytecode which can then run in a JVM
- **IronPython** – this one is written in C# and targets .Net (and mono) CLR
- **PyPy** – this one is written in RPython (which is itself a statically-typed subset of Python written in C that is specifically designed to write interpreters)
- and many more...

<https://wiki.python.org/moin/PythonImplementations>



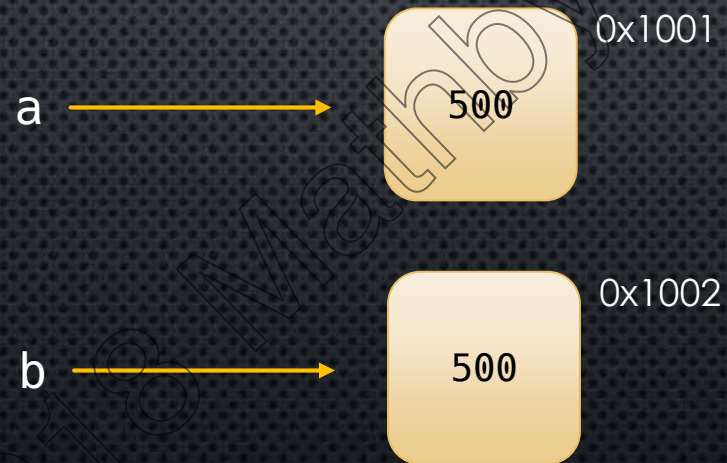
Earlier we saw:

```
a = 10  
b = 10
```



But look at this:

```
a = 500  
b = 500
```



In this case, although it would be safe for Python to create a shared reference, it does not!



# What is going on?

**Interning:** reusing objects on-demand

At startup, Python (CPython), **pre-loads** (caches) a global list of integers in the range [-5, 256]

Any time an integer is referenced in that range, Python will use the cached version of that object

**Singletons**

Optimization strategy – small integers show up often

When we write

```
a = 10
```

Python just has to point to the existing reference for 10

But if we write

```
a = 257
```

Python does not use that global list and a new object is created every time