PYTHON OPTIMIZATIONS

String Interning

Some strings are also automatically interned – but not all!

As the Python code is compiled, identifiers are interned

- variable names
- function names
- class names
- etc.

Identifiers:
- must start with _ or a letter
- can only contain _, letters and numbers

Some string literals may also be automatically interned:

- string literals that look like identifiers (e.g. `'hello_world'`)
- although if it starts with a digit, even though that is not a valid identifier, it may still get interned

But don't count on it!!

Why do this?

It’s all about (speed and, possibly, memory) optimization.

Python, both internally, and in the code you write, deals with lots and lots of dictionary type lookups, on string keys, which means a lot of string equality testing.

Let’s say we want to see if two strings are equal:

```
a = ‘some_long_string’     b = ‘some_long_string’
```

Using `a == b`, we need to compare the two strings character by character

But if we know that `‘some_long_string’` has been interned, then `a` and `b` are the same string if they both point to the same memory address

In which case we can use `a is b` instead – which compares two integers (memory address)

This is **much** faster than the character by character comparison

Not all strings are automatically interned by Python

But you can force strings to be interned by using the `sys.intern()` method.

```
import sys

a = sys.intern(‘the quick brown fox’)
b = sys.intern(‘the quick brown fox’)
```

`a is b` → True

much faster than `a == b`

When should you do this?

- dealing with a large number of strings that could have high repetition
  e.g. tokenizing a large corpus of text (NLP)
- lots of string comparisons

In general though, you do not need to intern strings
yourself. Only do this if you really need to.