Python에서의 GC

파이썬은 기본적으로 레퍼런스 카운팅(Reference Counting)을 바탕으로 GC를 수행하고 메모리를 관리한다.

레퍼런스 카운팅(Reference Counting)?

모든 객체는 참조 당할 때 이 레퍼런스 카운트를 증가시키고, 참조가 없어지면 이를 감소시킨다. 이 값이 0이 되면 객체가 메모리에서 해제된다. 해당 객체의 레퍼런스 카운트 값을 확인하는 코드는 다음과 같다.

```
sys.getrefcount(obj)
```

GC의 동작 원리

앞서 언급한 것처럼 GC는 레퍼런스 카운트를 기준으로 진행하는데, 좀 더 정확히 다루자면 세대와 그에 따른 임계값을 바탕으로 주기적으로 관리한다. 여기서 말하는 세대는 숫자가 클수록 오래된 객체이며, GC는 기본적으로 0세대, 즉, 비교적 최근에 생성된 객체에 대해 자주 GC를 수행하게 된다.

파이썬 GC에서 사용하는 세대는 0, 1, 2 세 가지고, 각각의 임계값은 다음 코드로 확인할 수 있고, 설정도 가능하다. 각각의 임계값을 초과하면 GC가 수행된다.

```
th0, th1, th2 = gc.get_threshold()
gc.set_threshold(1000, 100, 100)
```

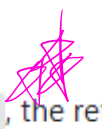
As I understand, sys.getrefcount() returns the number of references of an object, which "should" be 1 in the following case:

```
import sys,numpy
a = numpy.array([1.2,3.4])
print sys.getrefcount(a)
```

However, it turned out to be 2! So, if I:

```
del a
```

Will the "numpy.array([1.2,3.4])" object still be there (no garbage collection)?



'레퍼런스'에 의한 추가적인 참조 발생.

When you call `getrefcount()`, the reference is copied by value into the function's argument, temporarily bumping up the object's reference count. This is where the second reference comes from.

This is explained in the documentation:

> The count returned is generally one higher than you might expect, because it includes the (temporary) reference as an argument to `getrefcount()`.

As to your second question:

> If I "del a", will the "numpy.array([1.2,3.4])" object still be there (no garbage collection)?

By the time `getrefcount()` exits, the array's reference count will to back to 1, and a subsequent `del a` would release the memory.