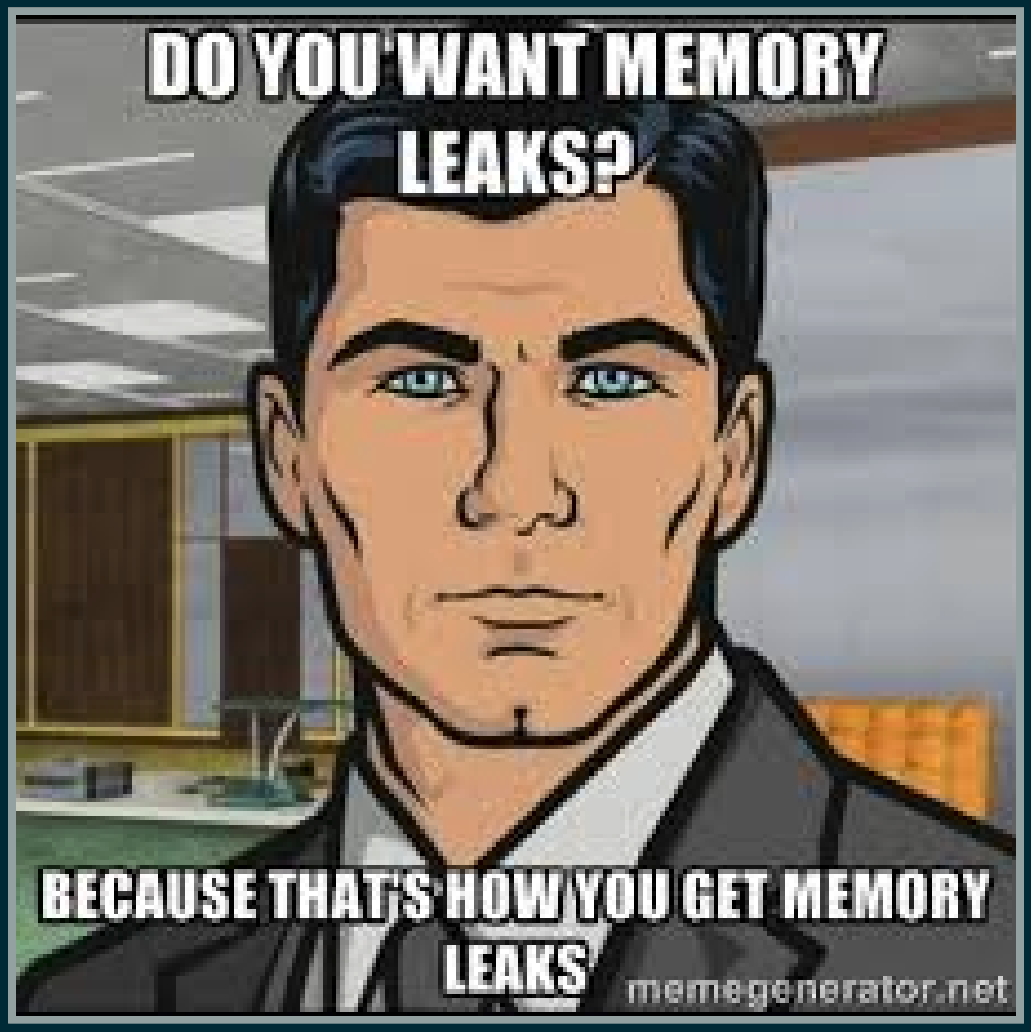
PYTHON MEMORY MANAGEMENT

@crlane

PyATL

January 14, 2016

BUT...PYTHON MANAGES ITS OWN MEMORY



COUNTING REFERENCES

```
/* PyObject_HEAD defines the initial segment of every PyObject. */
#define PyObject_HEAD                   PyObject ob_base;

#define PyObject_HEAD_INIT(type)        \
    { _PyObject_EXTRA_INIT              \
    1, type },

#define PyVarObject_HEAD_INIT(type, size)       \
    { PyObject_HEAD_INIT(type) size },

/* PyObject_VAR_HEAD defines the initial segment of all variable-size
 * container objects.  These end with a declaration of an array with 1
 * element, but enough space is malloc'ed so that the array actually
 * has room for ob_size elements.  Note that ob_size is an element count,
 * not necessarily a byte count.
 */
#define PyObject_VAR_HEAD      PyVarObject ob_base;
#define Py_INVALID_SIZE (Py_ssize_t)-1

/* Nothing is actually declared to be a PyObject, but every pointer to
 * a Python object can be cast to a PyObject*.  This is inheritance built
 * by hand.  Similarly every pointer to a variable-size Python object can,
 * in addition, be cast to PyVarObject*.
 */
typedef struct _object {
    _PyObject_HEAD_EXTRA
    Py_ssize_t ob_refcnt;
    struct _typeobject *ob_type;
} PyObject;
```

INSPECTING REFCOUNTS

```
>>> a = ['python', 'is', 'cool']
>>> hex(id(a))
'0x1088835a8'
>>> import ctypes
>>> def refcount(obj):
        return ctypes.c_size_t.from_address(id(obj))
```

INSPECT A MEMORY ADDRESS

```
>>> a_ref_count = refcount(a)
>>> a_ref_count
c_ulong(1L)
```

BREAK REFERENCE COUNTING

```
>>> b = a
>>> a_ref_count
c_ulong(2L)
>>> a_ref_count.value = 1
>>> a_ref_count
c_ulong(1L)
```

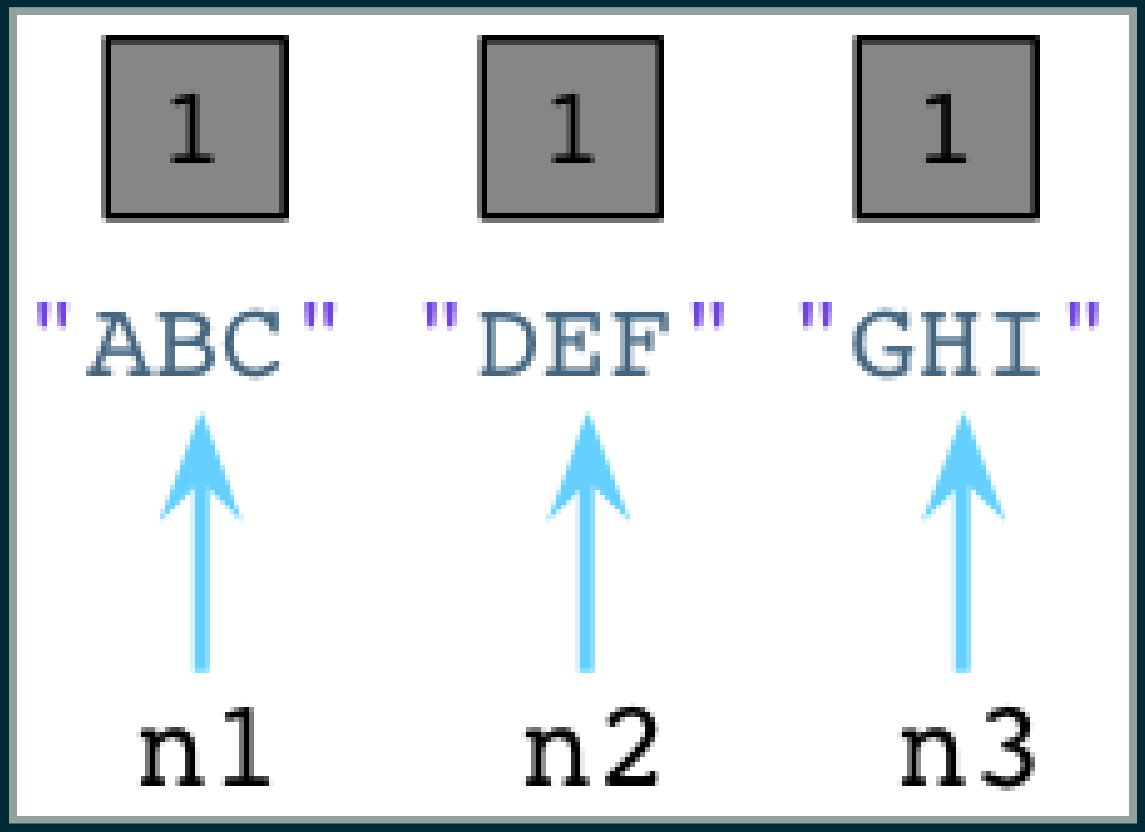
DIRTY, DIRTY HACKS

```
>>> a
['python', 'is', 'cool']
>>> del b
c = ['please', 'do', 'not', 'do', 'this']
>>> a
['please', 'do', 'not', 'do', 'this']
>>> hex(id(c))== hex(id(a))
True
```

OBJECT LIFECYCLE

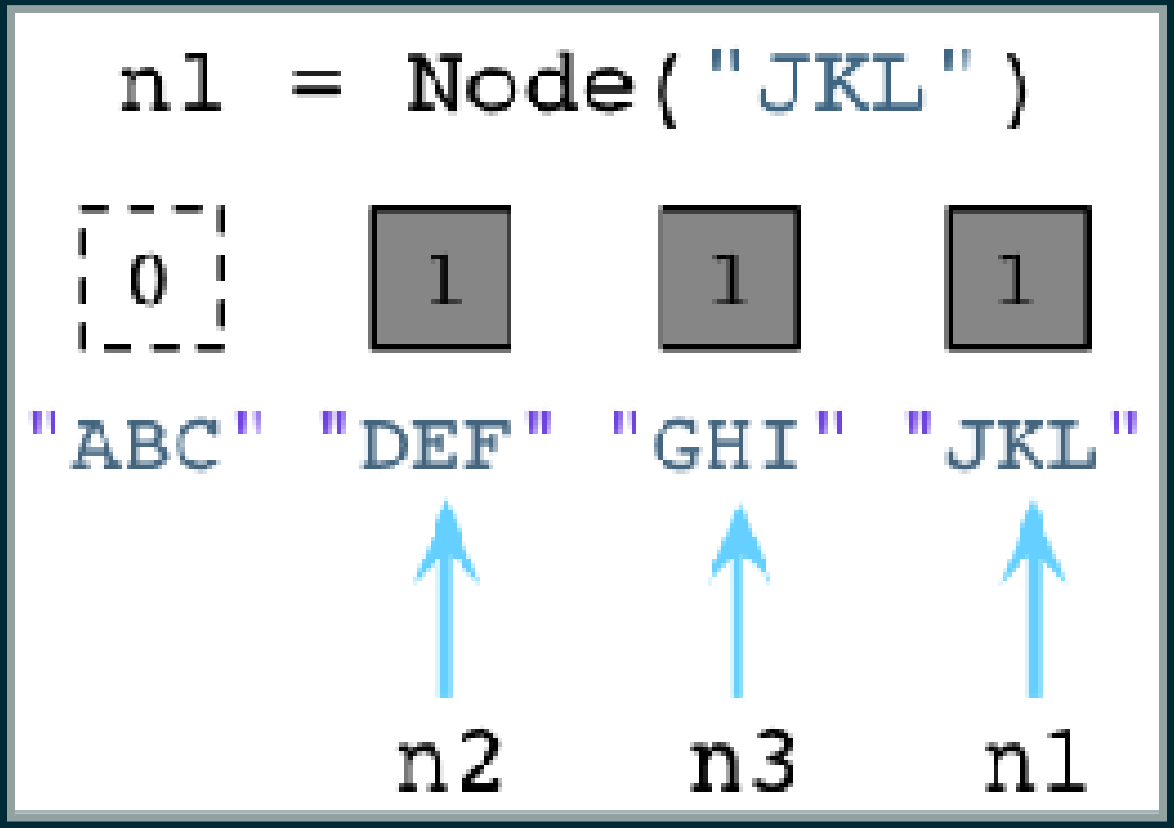
```
class Node(object):
    def __init__(self, value):
        self.value = value

n1 = Node('ABC')
n2 = Node('DEF')
n3 = Node('GHI')
```

OBJECT LIFECYCLE



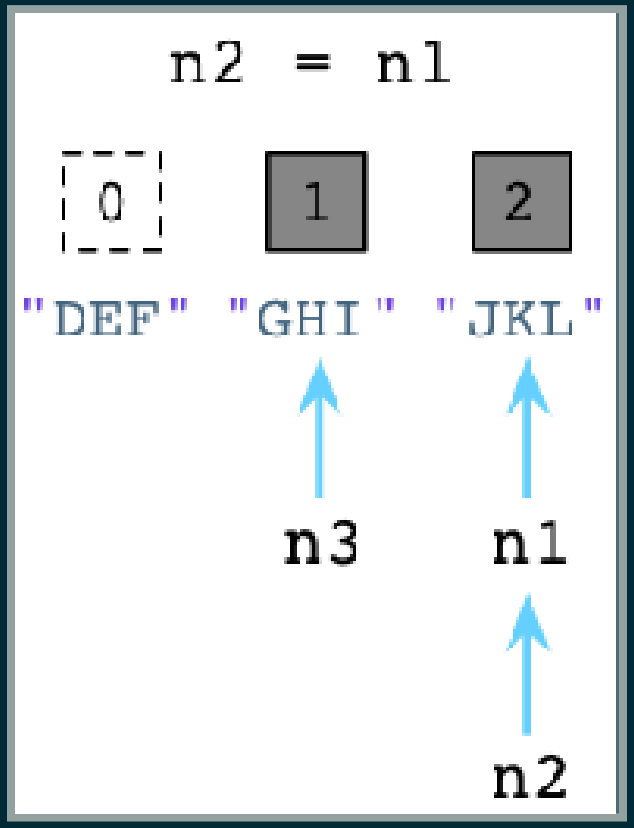
http://patshaughnessy.net/2013/10/24/visualizing-garbage-collection-in-ruby-and-python

REFCOUNT 0



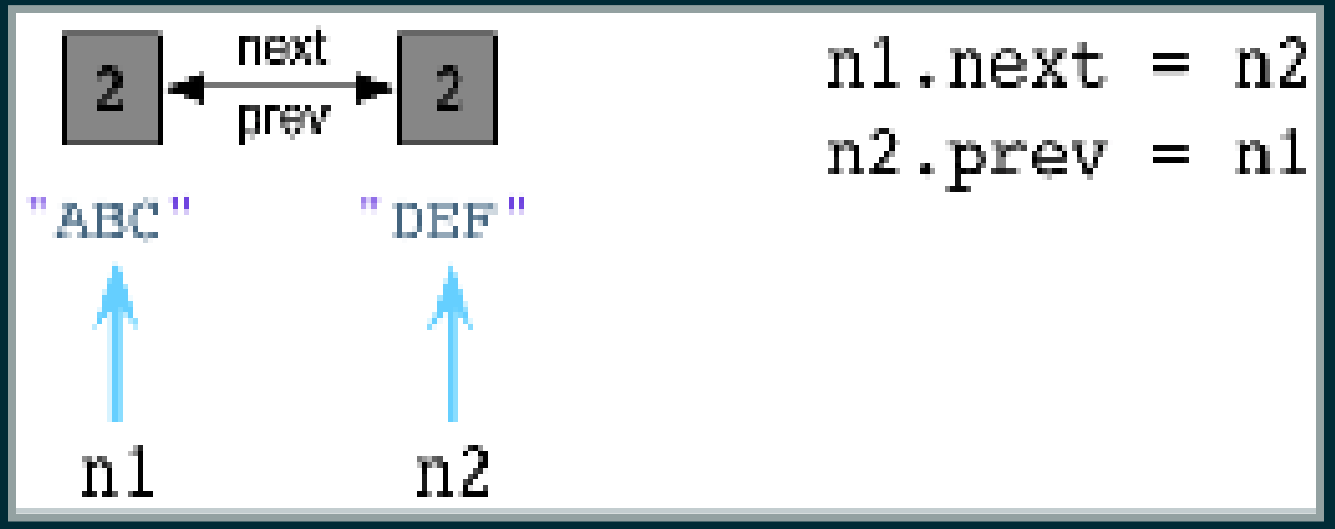
http://patshaughnessy.net/2013/10/24/visualizing-garbage-collection-in-ruby-and-python

REASSIGNING LABELS



http://patshaughnessy.net/2013/10/24/visualizing-garbage-collection-in-ruby-and-python

REFERENCE CYCLES



http://patshaughnessy.net/2013/10/30/generational-gc-in-python-and-ruby

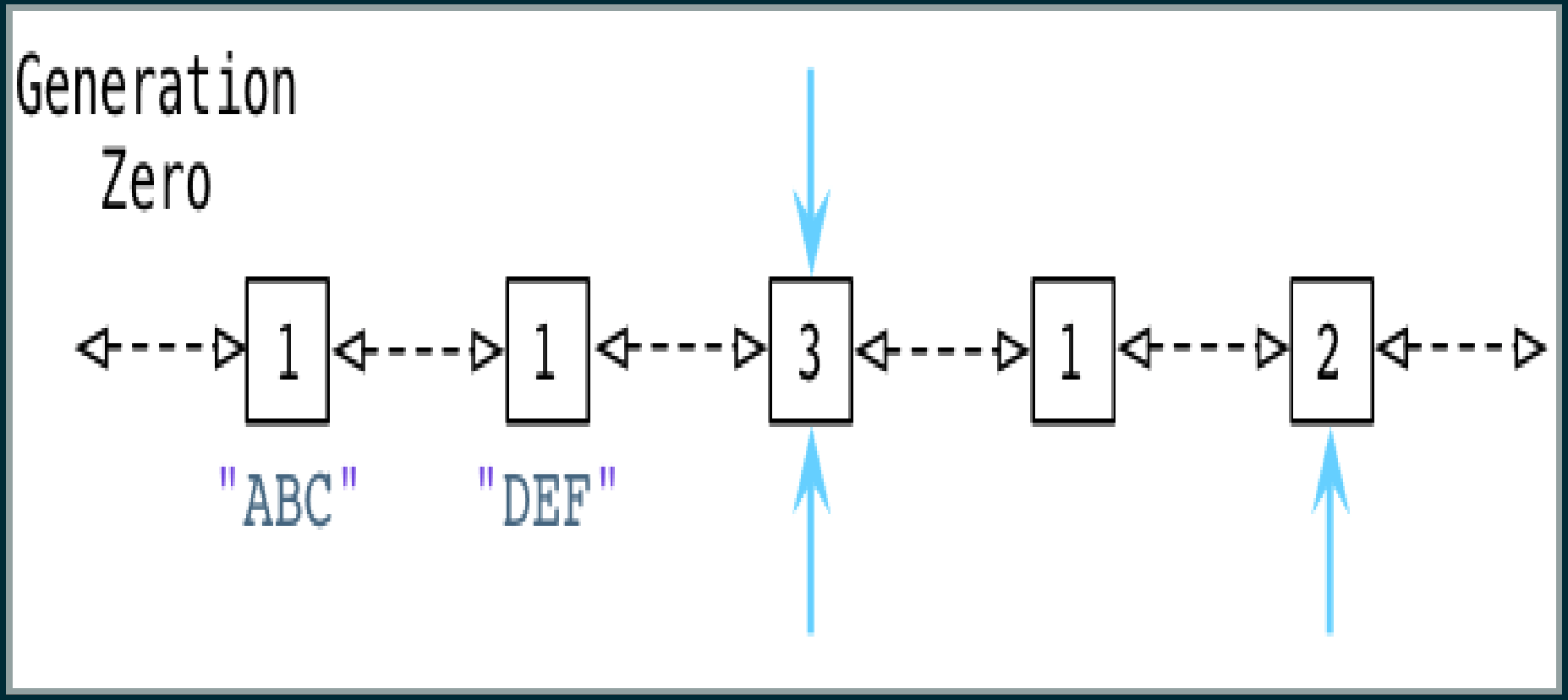
GARBAGE COLLECTION

http://www.heiloftexas.com/productsItem.asp?intProductID=DuraPack%20Python

KEY POINTS

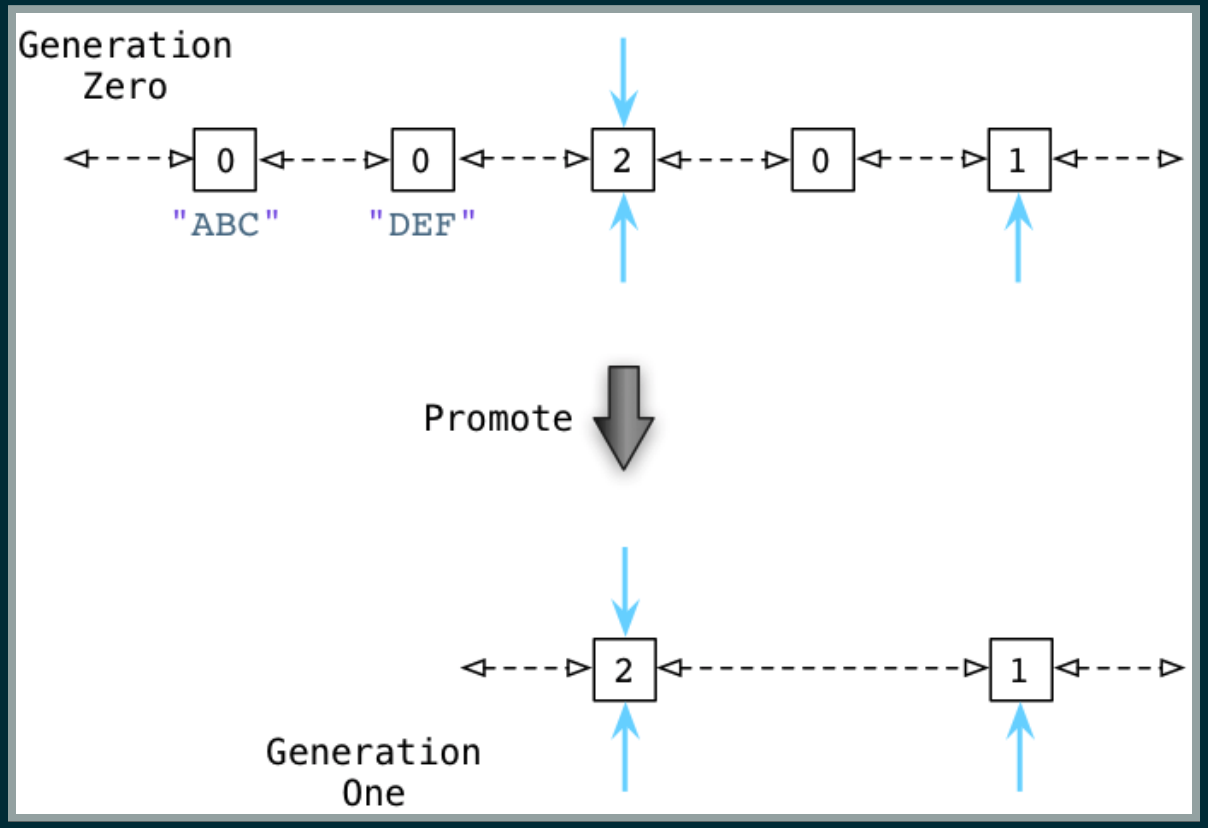
- Generational garbage collector
- Most objects are short lived
- Three linked lists of allocated objects
- Tunable frequency parameters

GENERATION 0



http://patshaughnessy.net/2013/10/30/generational-gc-in-python-and-ruby

GENERATION 1



http://patshaughnessy.net/2013/10/30/generational-gc-in-python-and-ruby

GC IS NOT FREE

```
gc: collecting generation 0...
gc: objects in each generation: 1544 1594 36919
gc: done, 628 unreachable, 0 uncollectable, 0.0003s elapsed.
gc: collecting generation 1...
gc: objects in each generation: 924 1657 36919
gc: done, 1504 unreachable, 0 uncollectable, 0.0006s elapsed.
gc: collecting generation 2...
gc: objects in each generation: 1051 0 37801
gc: done, 550 unreachable, 0 uncollectable, 0.0139s elapsed.
```

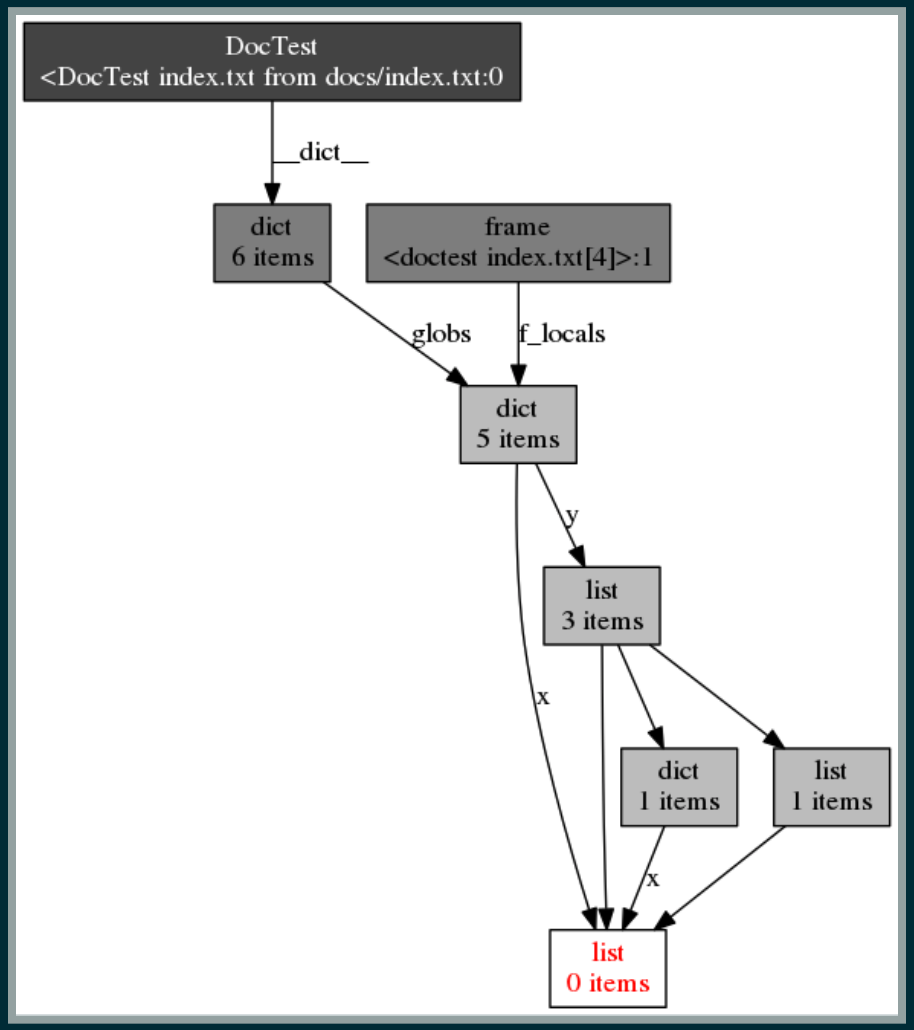
GC

- standard library module
- provides visibility into garbage collection

GC

```
>>> import gc
>>> gc.get_referrers(obj)
...
>>> gc.get_referents(obj)
>>> gc.collect([generation])
...
>>> gc.get_threshold()
(700, 10, 10)
>>> gc.set_threshold(100, 5, 5)
>>> gc.set_debug(gc.DEBUG_LEAK) # DEBUG_STATS, et al.
```

OBJGRAPH



https://mg.pov.lt/objgraph/_images/sample-backref-graph.png

OTHER CONSIDERATIONS

- `__del__`
- some types (e.g., int & strings) are special
- internal memory pools
- heap fragmentation

REFERENCES

- http://blog.hakril.net/articles/0-understanding-python-by-bre
- http://patshaughnessy.net/2013/10/24/visualizing-garbage-c
  ruby-and-python
- http://www.evanjones.ca/memoryallocator/
- http://pymotw.com/2/gc/
- http://mg.pov.lt/blog/hunting-python-memleaks
- https://mg.pov.lt/objgraph/
- http://deeplearning.net/software/theano/tutorial/python-mem
  management.html
- http://www.slideshare.net/delimitry/python-gc
- http://www.digi.com/wiki/developer/index.php/Python_Garba