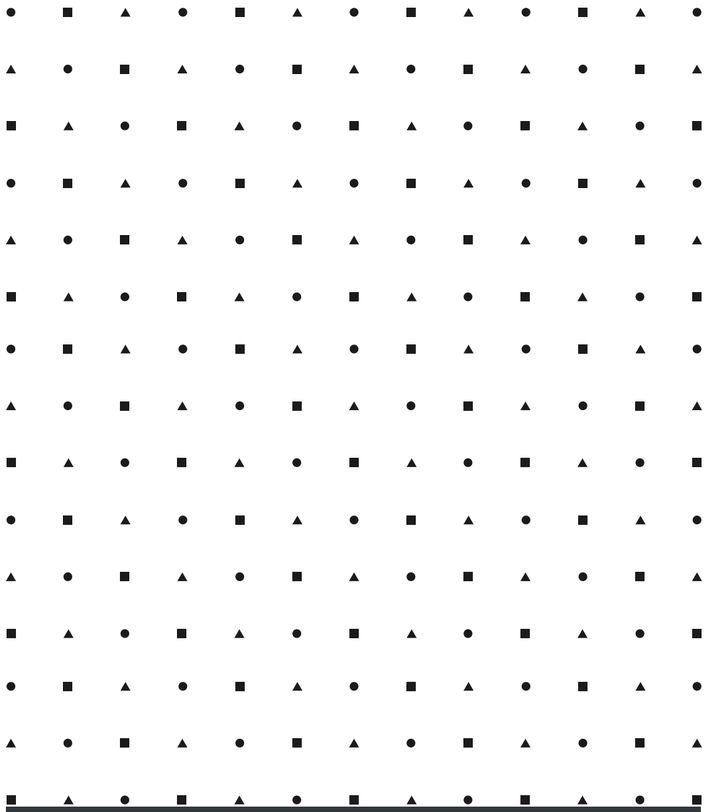


Python Memory Understanding

By Nisarg Shah

Nisarg Shah

Crazy Developer - Love creating Bugs!

HELLO!

Undergrad CS Student

Software Developer intern @Fountain9

CONNECT WITH ME!

iamnisarg.in

nisarg1499

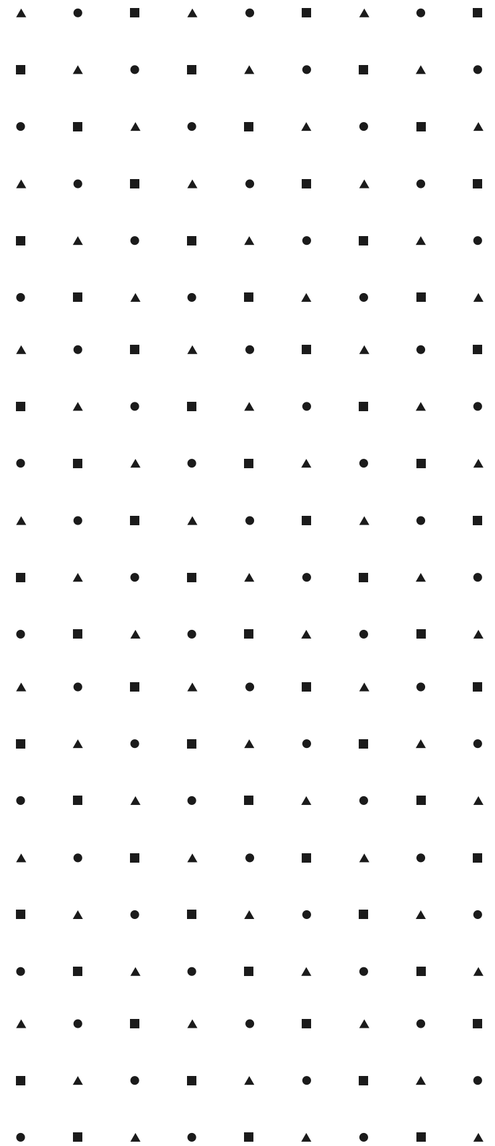
nisargshah14

nisshah1499@gmail.com

Today's Talk

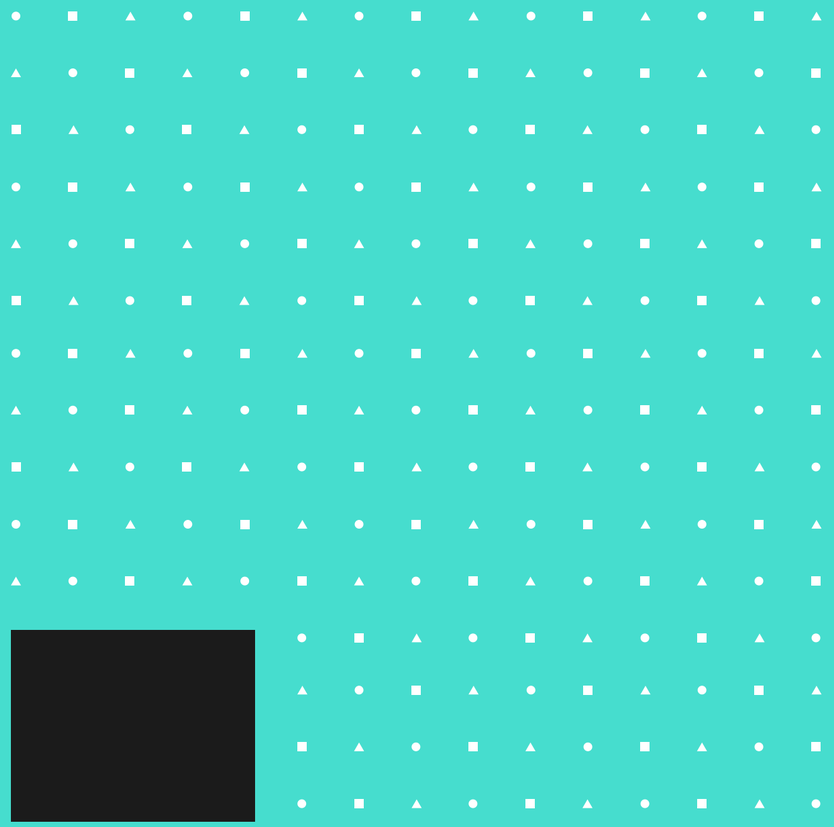
What will be covered?

- Python Objects
- Memory Storage
- Garbage Collection
- Reference counting and Circular reference
- Working of GC Algorithms



Python version : 3.8.5

Heard of only Objects



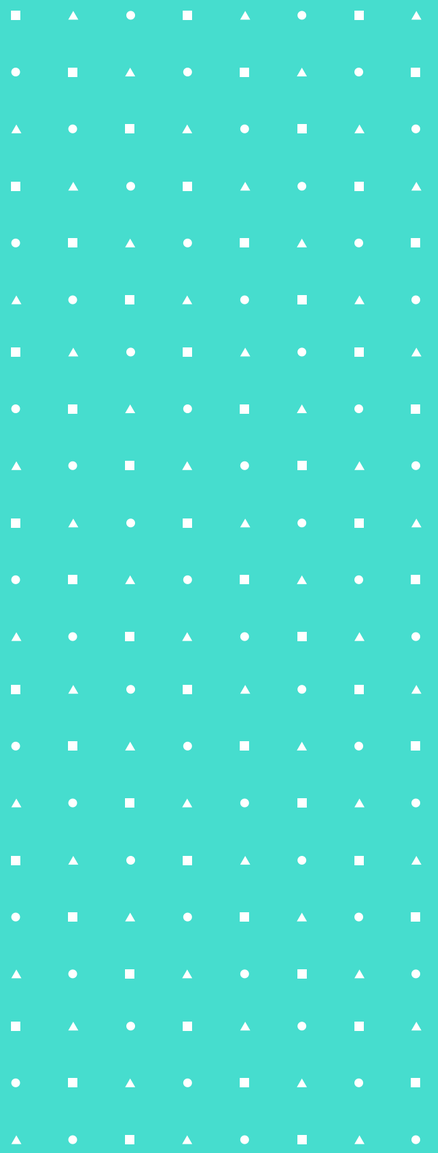
- Everything has an unique ID, type, value and reference count

- CPython
  - id()
  - type()

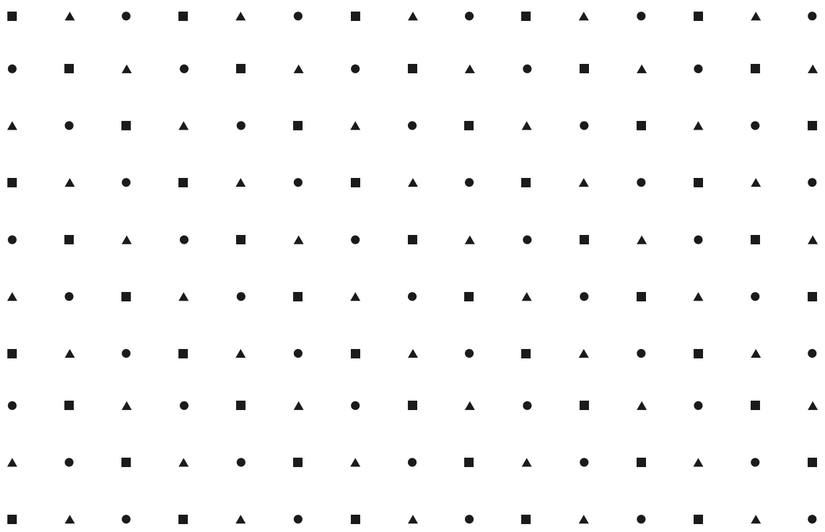
```
>>> variable = 9
>>> id(variable)
9785152
>>> type(variable)
<class 'int'>
```

Memory Storage

- Heap
  - Objects
  - Instances
- Stack
  - Methods
- Python Manager : PyMalloc
  - Memory allocation
  - Speeds up memory operations



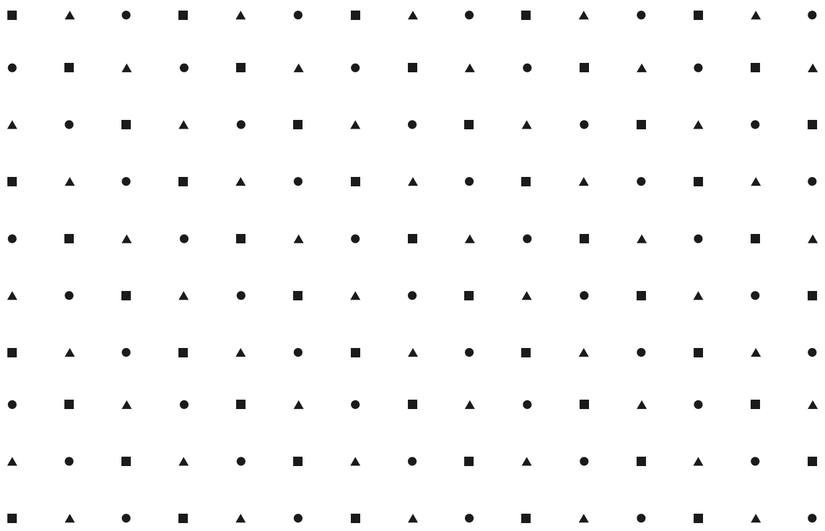
- Hundreds of objects
- Long running python processes
- Lot's of memory usage
- If mistake in memory de-allocation, program can be crashed



Let's learn about

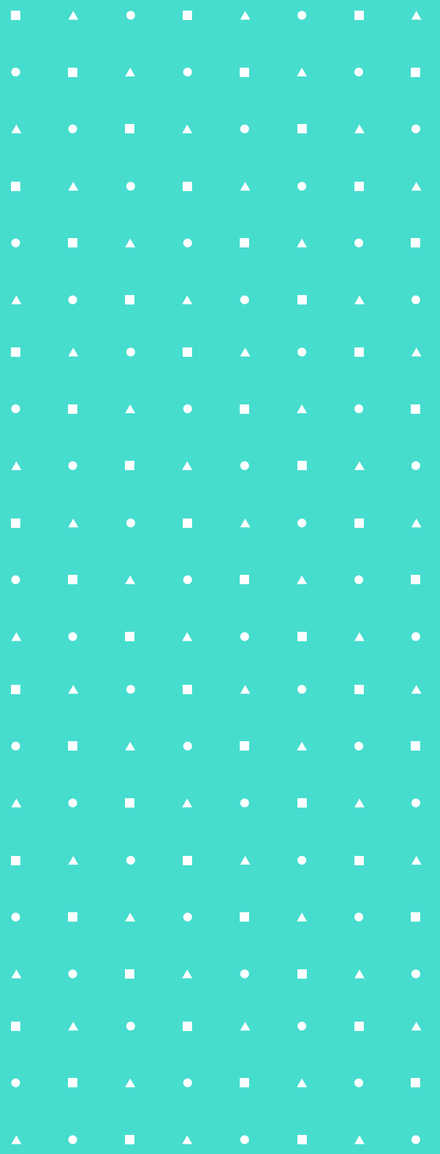
Garbage collection

- Tracks which objects to be deallocated
- Automatic deallocation
- Programmer relief



GC algorithms

- Reference counting algo
  - Easy
  - Efficient
  - Straightforward
- Generational GC
  - some what tricky
  - optional, can be manually triggered

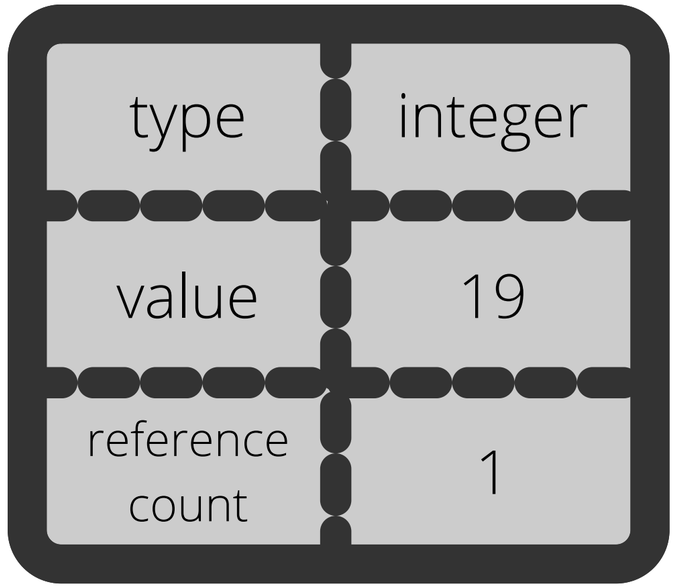


Reference count

Python object

All properties are automatically detected by python

```
>>> a = 19
>>> id(a)
9785472
```

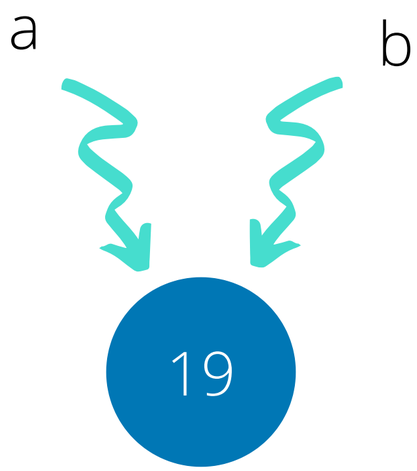


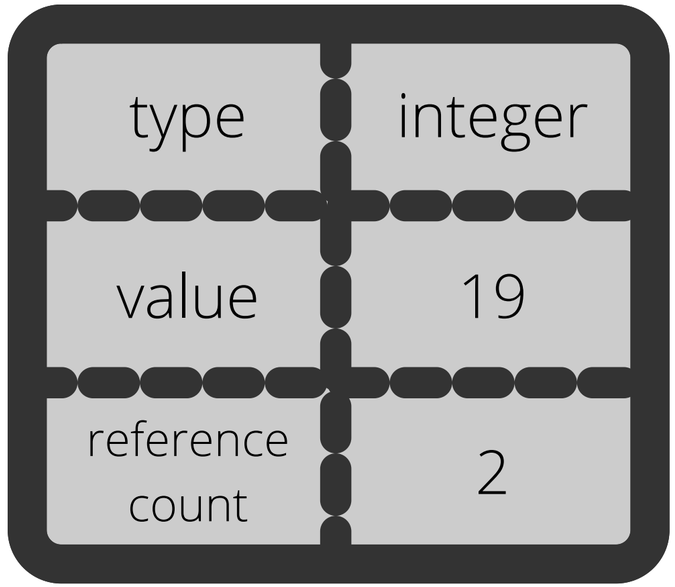
Count++

b = a
c = 19

Get count :
import sys
sys.getrefcount(object)

```
>>> id(a)
9785472
>>> b = a
>>> id(b)
9785472
>>> c = 19
>>> id(c)
9785472
```





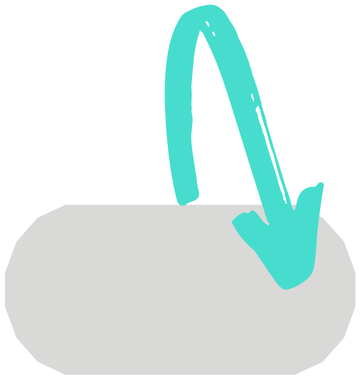
Lists!

Reference count works same with lists
But if you use copy, it will be another object

```
>>> l1 = [1,2,3]
>>> id(l1)
140602549677440
>>> l2 = l1
>>> id(l2)
140602549677440
>>> import copy
>>> l3 = copy.copy(l1)
>>> id(l3)
140602549668096
>>> l1 = [1,3,4]
>>> id(l1)
140602549657088
```

Let's talk about

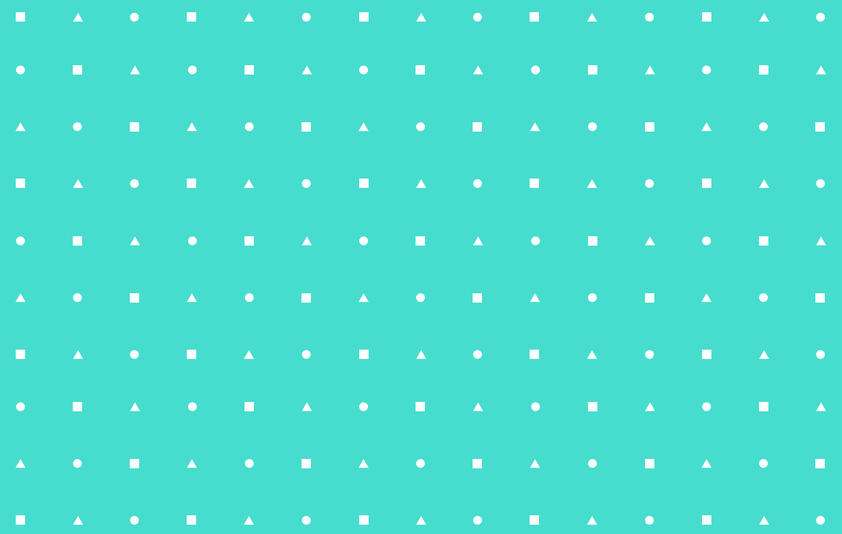
Circular reference



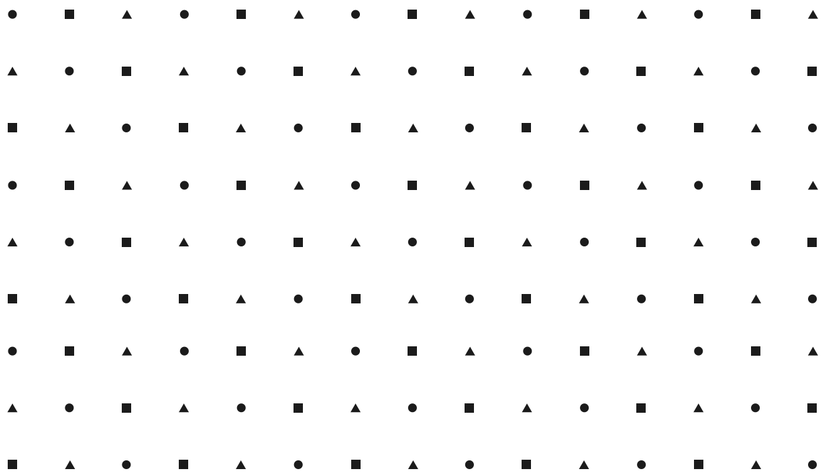
```
################

list1 = []
list1.append(l1)

################

Object1 = {}
Object2 = {}
Object1["second"] = Object2
Object1["first"] = Object1
```

Ref counting GC algo

- reference count == 0
- Objects referenced in another object
  - l1 = [1,2,3]
  - l2 = [4,5,6] and append l1
- Global variables count != 0



- Variables defined inside block ?
- Function execution is completed ?
- Manually delete the object?
  - del() method

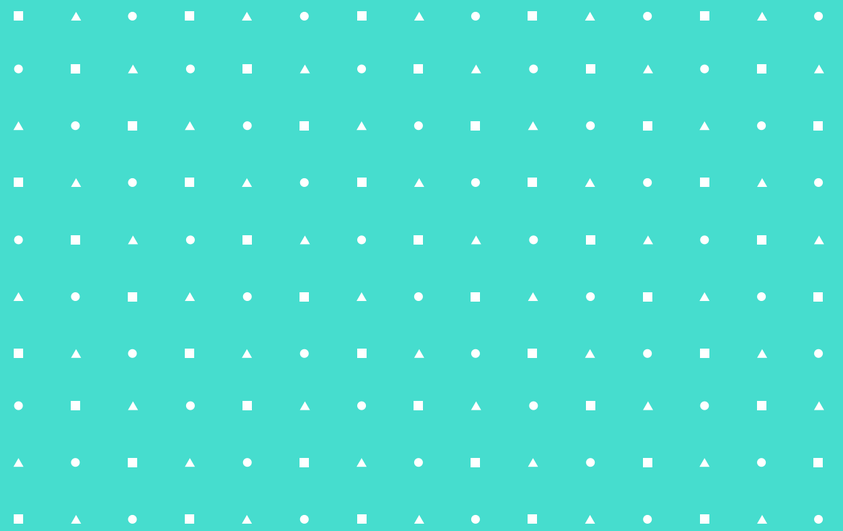


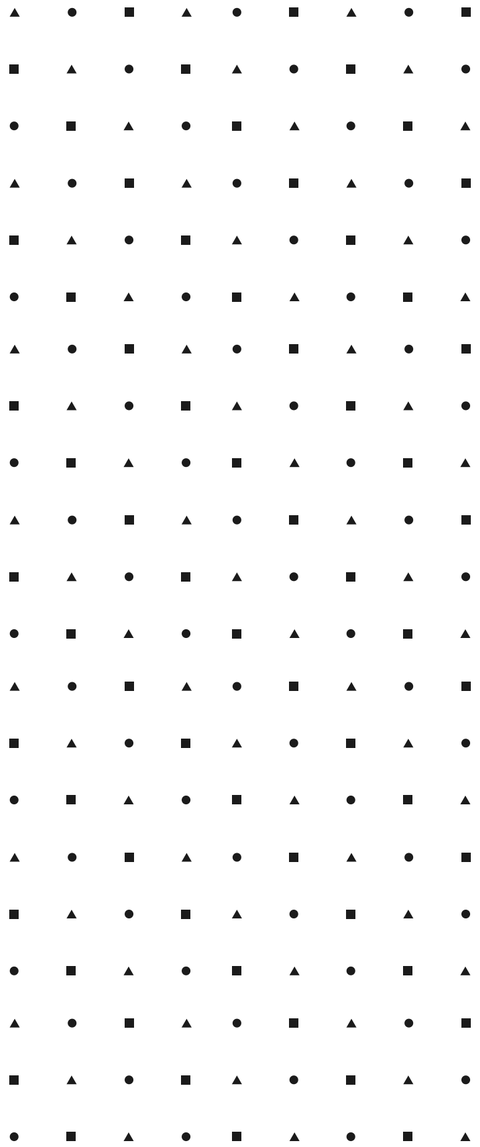
```
def foo(tempName):
    tempName = tempName + " Shah"
    print("My name is ", tempName)


if __name__ == "__main__":
    myName = "Nisarg"
    foo(myName)
```

Issues with RC algorithm

- Weak algorithm
- Can't detect Circular reference
- Memory and performance issues

But, RC is easy and objects are deleted immediately when they are of no use!

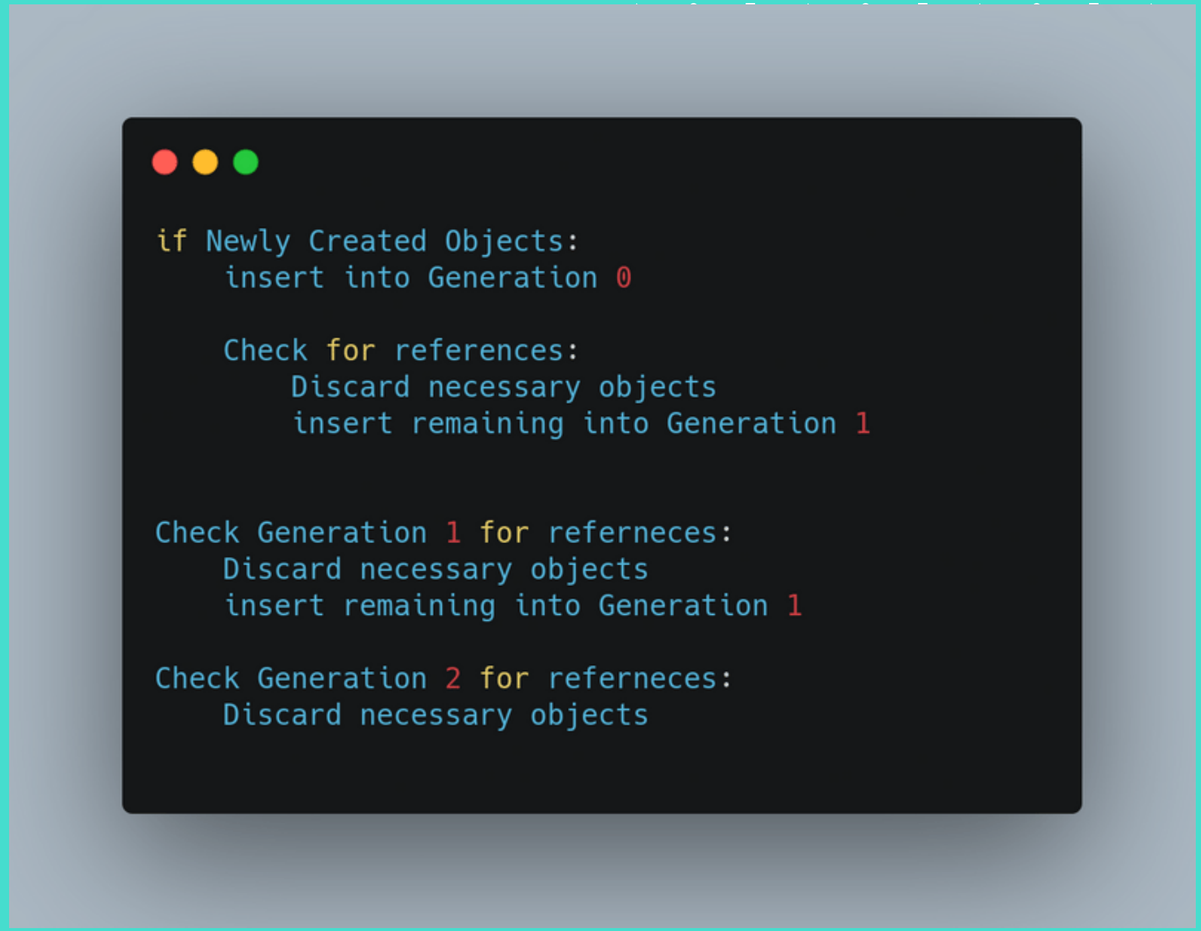




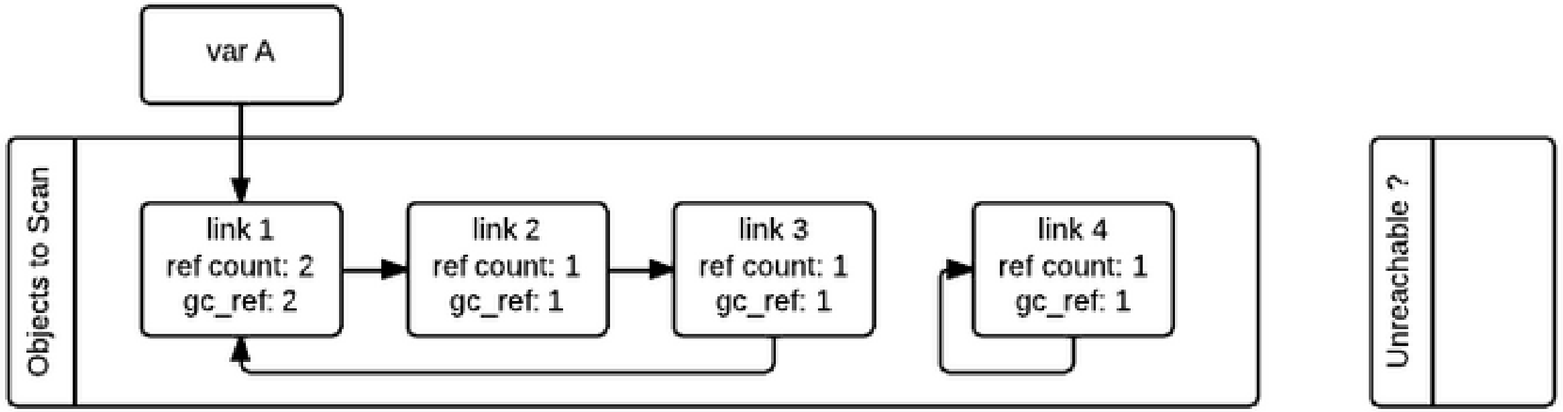
Generational GC

- Detects cycles
- Trace based garbage algorithm
- Delete unreached/unused objects

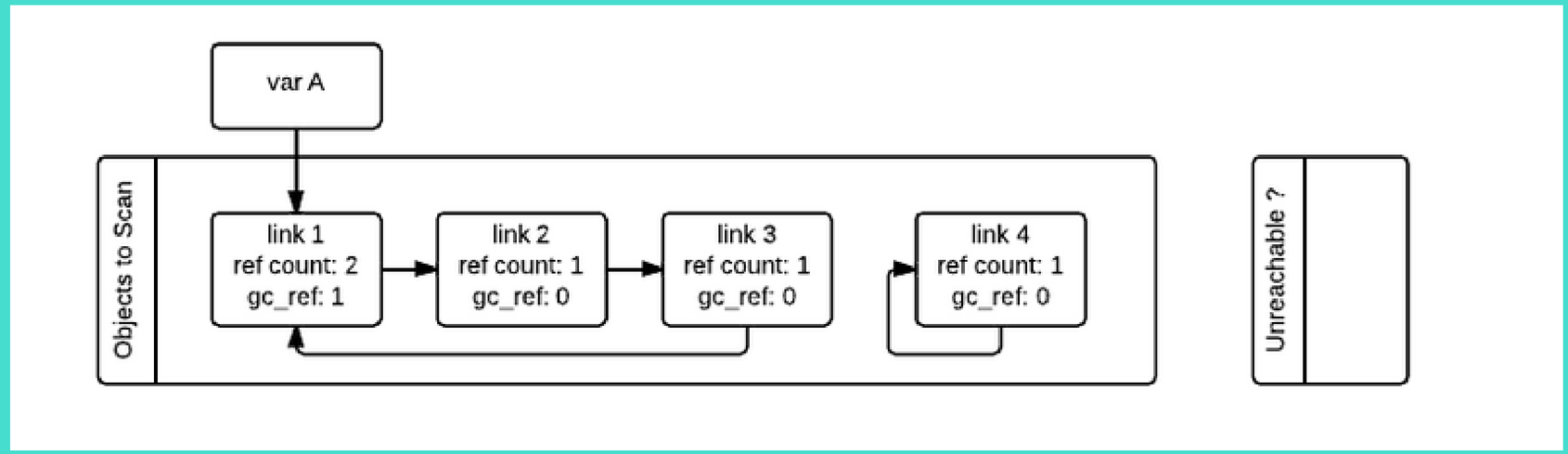
Not in real time

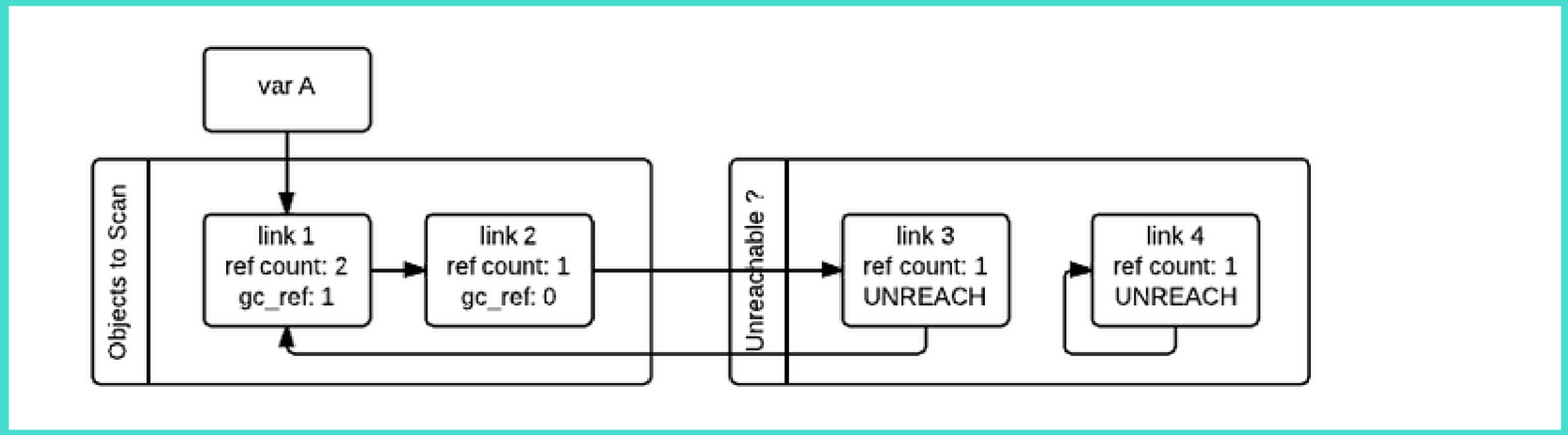


Overview of working

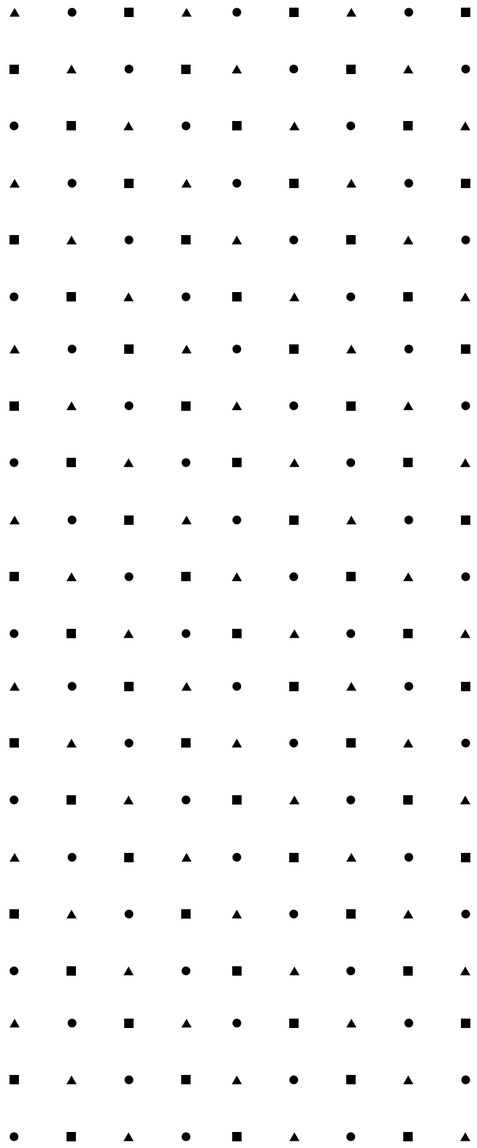


Taken from : https://devguide.python.org/garbage_collector/





Taken from : https://devguide.python.org/garbage_collector/



Want to know more?

- Design of CPython’s Garbage Collector
- https://docs.python.org/3.6/library/gc.html
  - gc.garbage()
  - gc.collect()
- Python Memory Model blogs
- Tracing Garbage Collection : Wikipedia

Thank you

CONNECT WITH ME!

iamnisarg.in

nisarg1499

nisargshah14

nisshah1499@gmail.com

-Nisarg Shah