

# **BASICS OF MEMORY MANAGEMENT IN PYTHON**



**Nina Zakharenko**



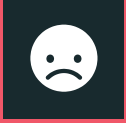
# WHY SHOULD YOU CARE?

Knowing about memory management helps you write more efficient code.



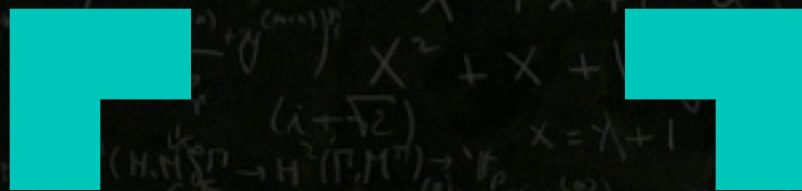
# WHAT WILL YOU GET?

- Vocabulary
- Basic Concepts
- Foundation

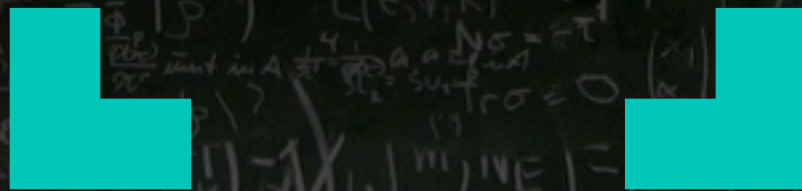


# WHAT WON'T YOU GET?

You won't be an expert at the end of this talk.



# WHAT'S A VARIABLE?

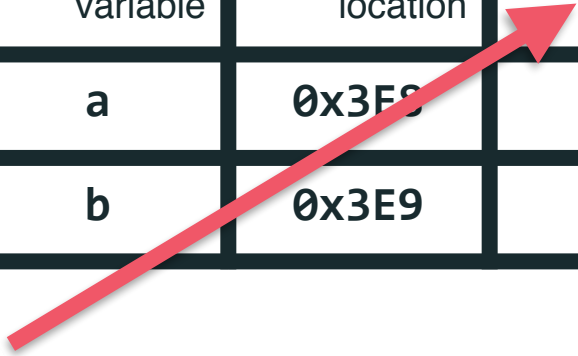


# What's a C-style variable?

```
/* c code */  
int a = 5;  
int b = 5;
```

Memory

variable	location	Value
a	0x3E8	101
b	0x3E9	101



These values live in a **fixed** size bucket.

Can only hold same-sized data, or an overflow occurs.

# What's a C-style variable?

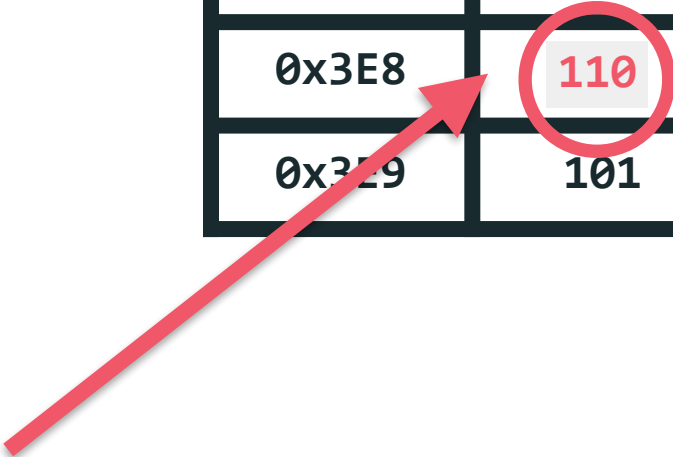
Later...

```
a = 6;
```

The data in this  
memory location is  
**overwritten.**

Memory

location	Value
0x3E8	110
0x3E9	101



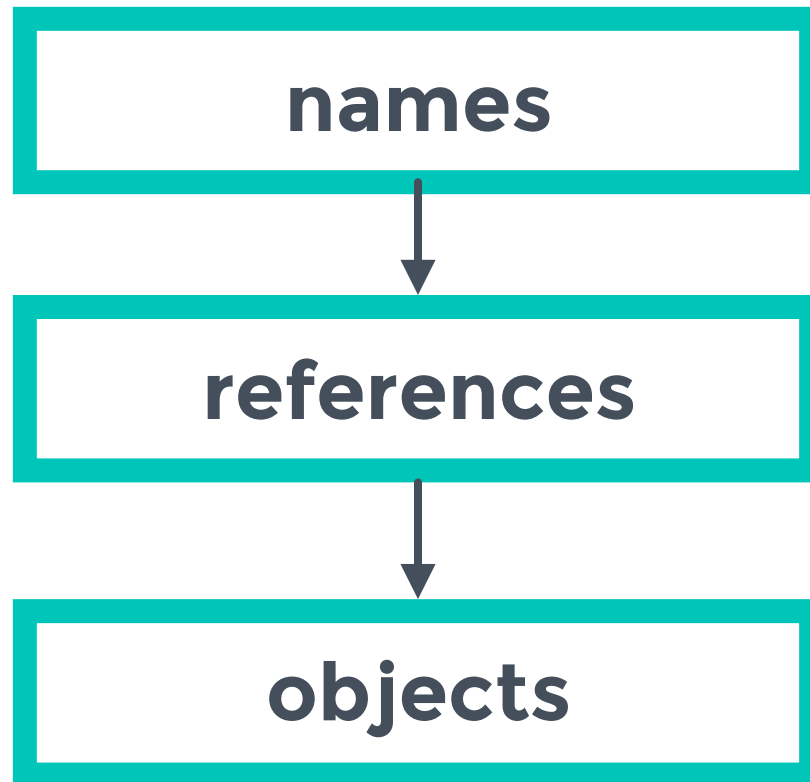


**PYTHON  
HAS NAMES,  
NOT  
VARIABLES**



# How are python objects stored in memory?

---





# **A name is just a label for an object.**

In python, each object can have  
lots of names.

# Different Types of Objects

---

## Simple

- numbers
- strings

## Containers

- dict
- list
- user defined-classes



# **What is a reference?**

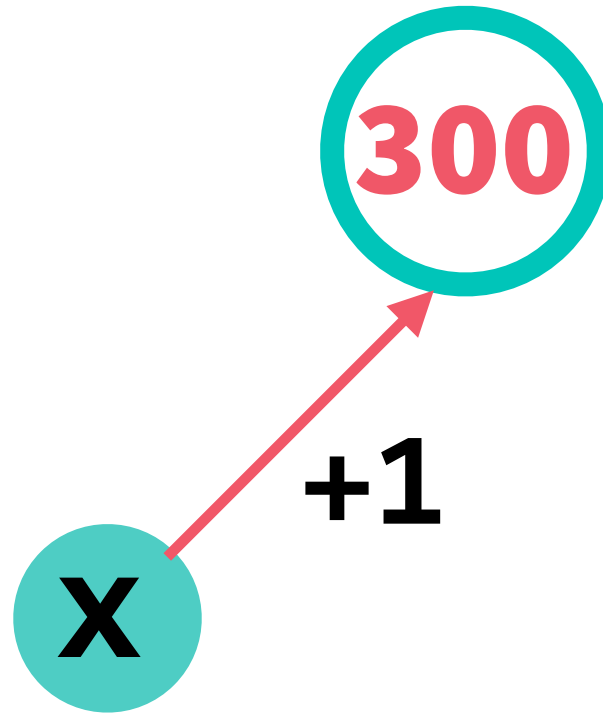
**A name or a container object pointing at another object.**



**What is a  
reference count?**

# How can we increase the ref count?

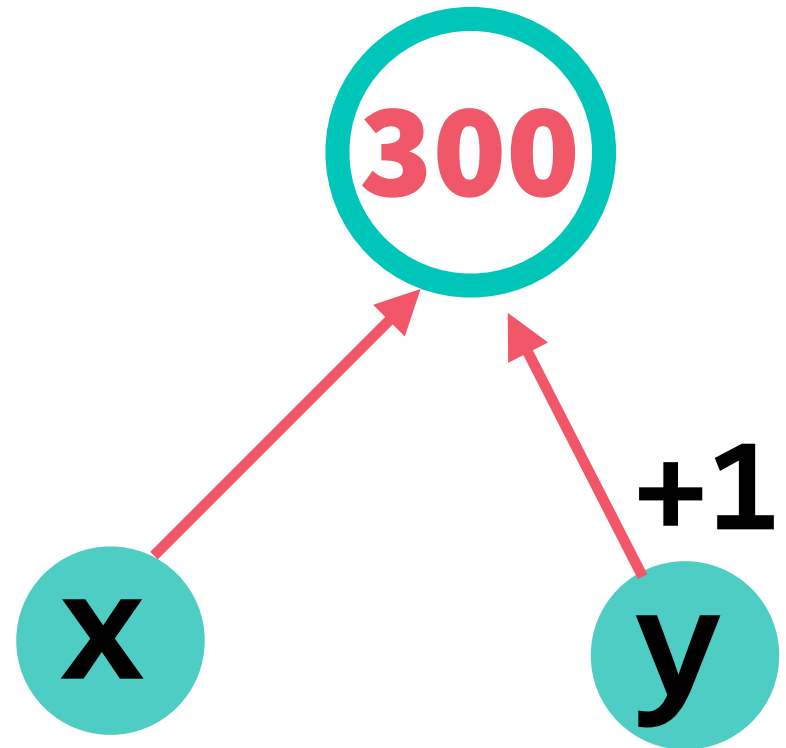
`x = 300`



references: 1

# How can we increase the ref count?

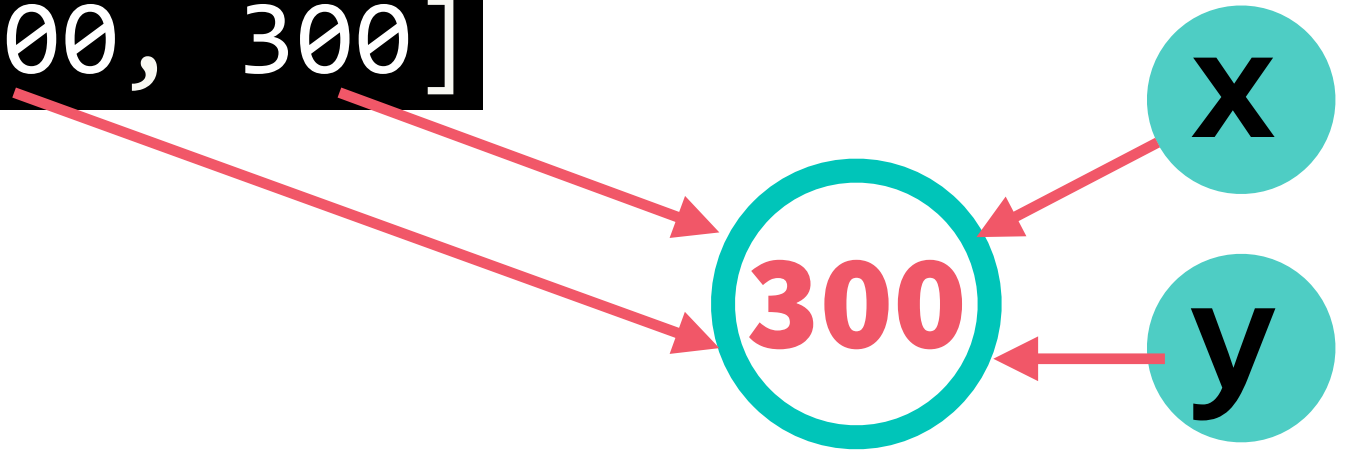
```
x = 300  
y = 300
```



references: 2

# How can we increase the ref count?

`z = [300, 300]`

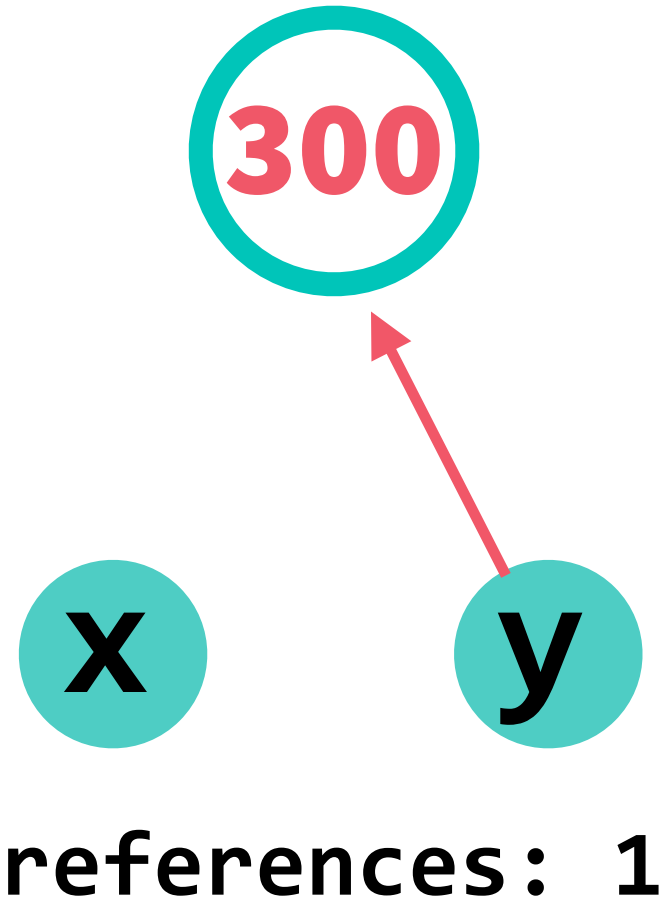


references: 4



## Decrease Ref Count - del

```
x = 300  
y = 300  
  
del x
```



# What does `del` do?

---

The `del` statement **doesn't delete** objects.

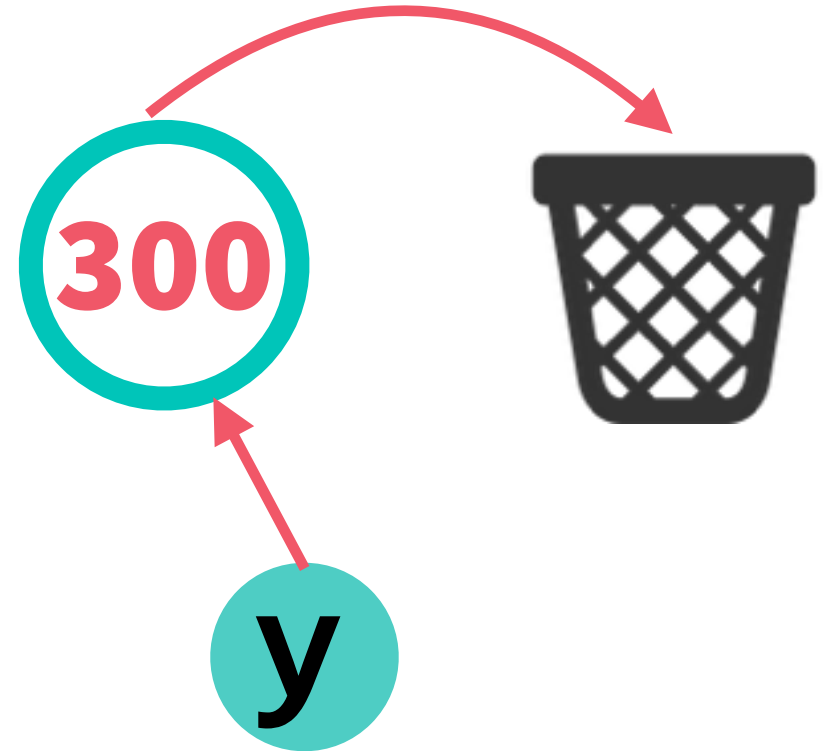
It:

- removes that name as a reference to that object
- reduces the ref count by 1

# Decrease Ref Count - Change Reference

```
x = 300  
y = 300
```

```
y = None
```



references: 0

# Decrease Ref Count - Going out of Scope

---

ref count **+1**

```
def print_word():  
    word = 'Seven'  
    print('Word is ' + word)
```

**'seven' is out of  
scope.  
ref count -1**

```
print_word()
```

# local vs. global namespace

---

- If refcounts decrease when an object goes out of scope, what happens to objects in the global namespace?
- Never go out of scope! Refcount never reaches 0.
- Avoid putting large or complex objects in the global namespace.



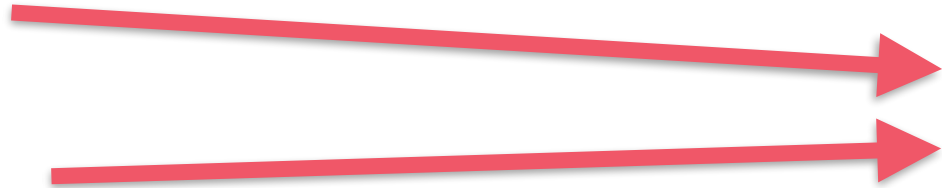
# Every python object holds 3 things

- Its type
- Its value
- A reference count

# Names      References

**x**

**y**



## PyObject

type	integer
refcount	2
value	300

```
x = 300  
y = 300
```

**\* don't try this in an interactive environment (REPL)**

```
print( id(x) )  
> 28501818
```

```
print( id(y) )  
> 28501818
```

```
print x is y  
> True
```





**GARBAGE  
COLLECTION**



# **What is Garbage Collection?**

**A way for a program to automatically release memory when the object taking up that space is no longer in use.**

# Two Main Types of Garbage Collection

---



**Reference  
Counting**

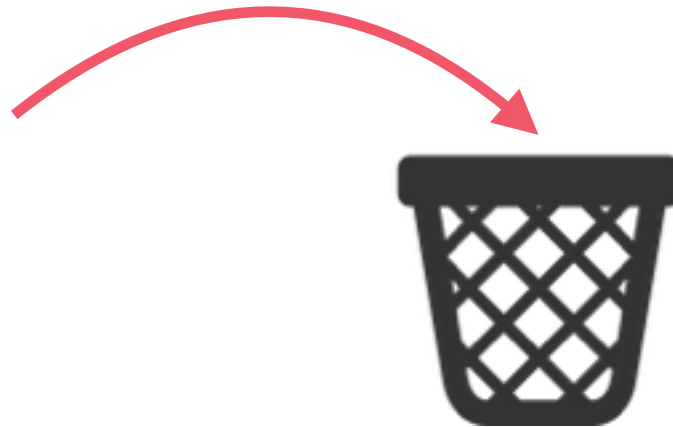
**Tracing**

# How does reference counting garbage collection work?

---

## Add and Remove References

**RefCount Reaches 0**



**Cascading Effect**

# Reference Counting Garbage Collection

---

## The Good

- **Easy to Implement**
- When refcount is 0, objects are **immediately deleted.**

## The Bad

- **space overhead** - reference count is stored for every object
- **execution overhead** - reference count changed on every assignment

# Reference Counting Garbage Collection

---

## The Ugly

- Not generally thread safe
- Reference counting doesn't detect cyclical references

# Cyclical References By Example

---

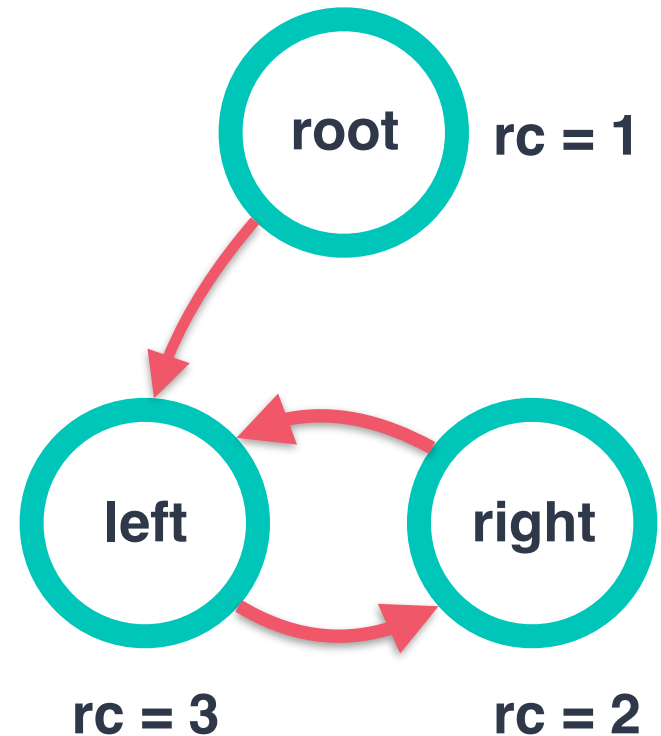
```
class Node:
    def __init__(self, value):
        self.value = value

    def next(self, next):
        self.next = next
```

# What's a cyclical reference?

```
root = Node('root')  
left = Node('left')  
right = Node('right')
```

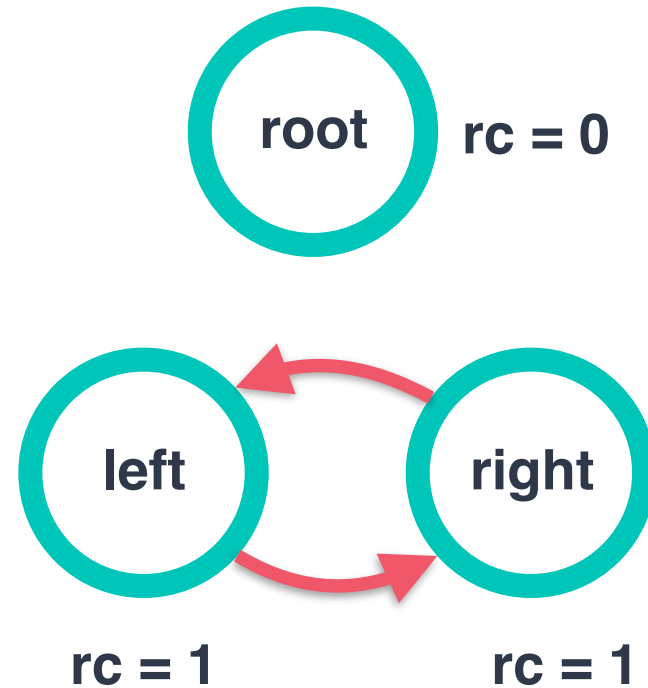
```
root.next(left)  
left.next(right)  
right.next(left)
```





# What's a cyclical reference?

```
del root  
del node1  
del node2
```





**Reference counting alone will not  
garbage collect objects with **cyclical  
references.****

# Two Main Types of Garbage Collection

---

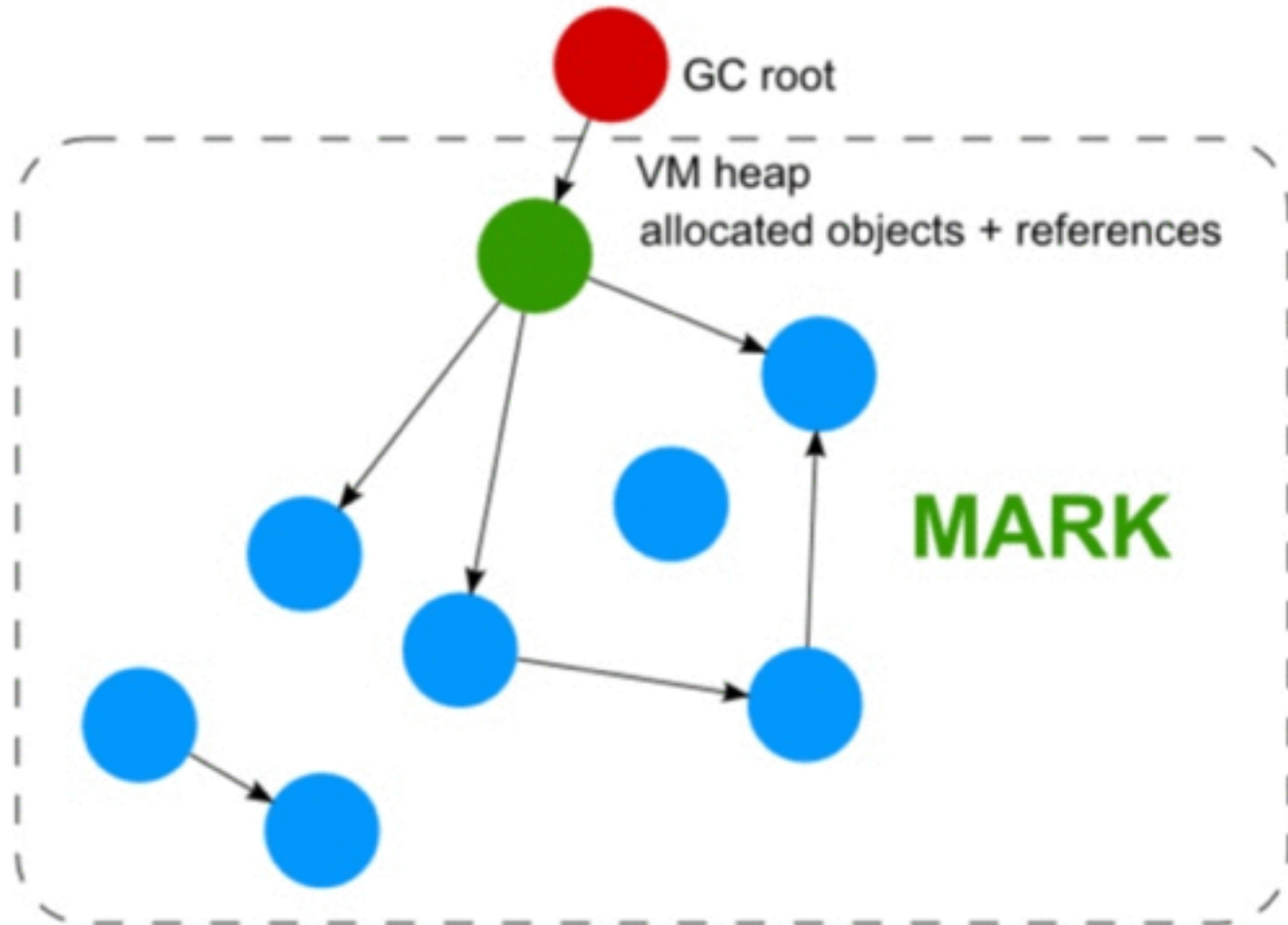


**Reference  
Counting**

**Tracing**

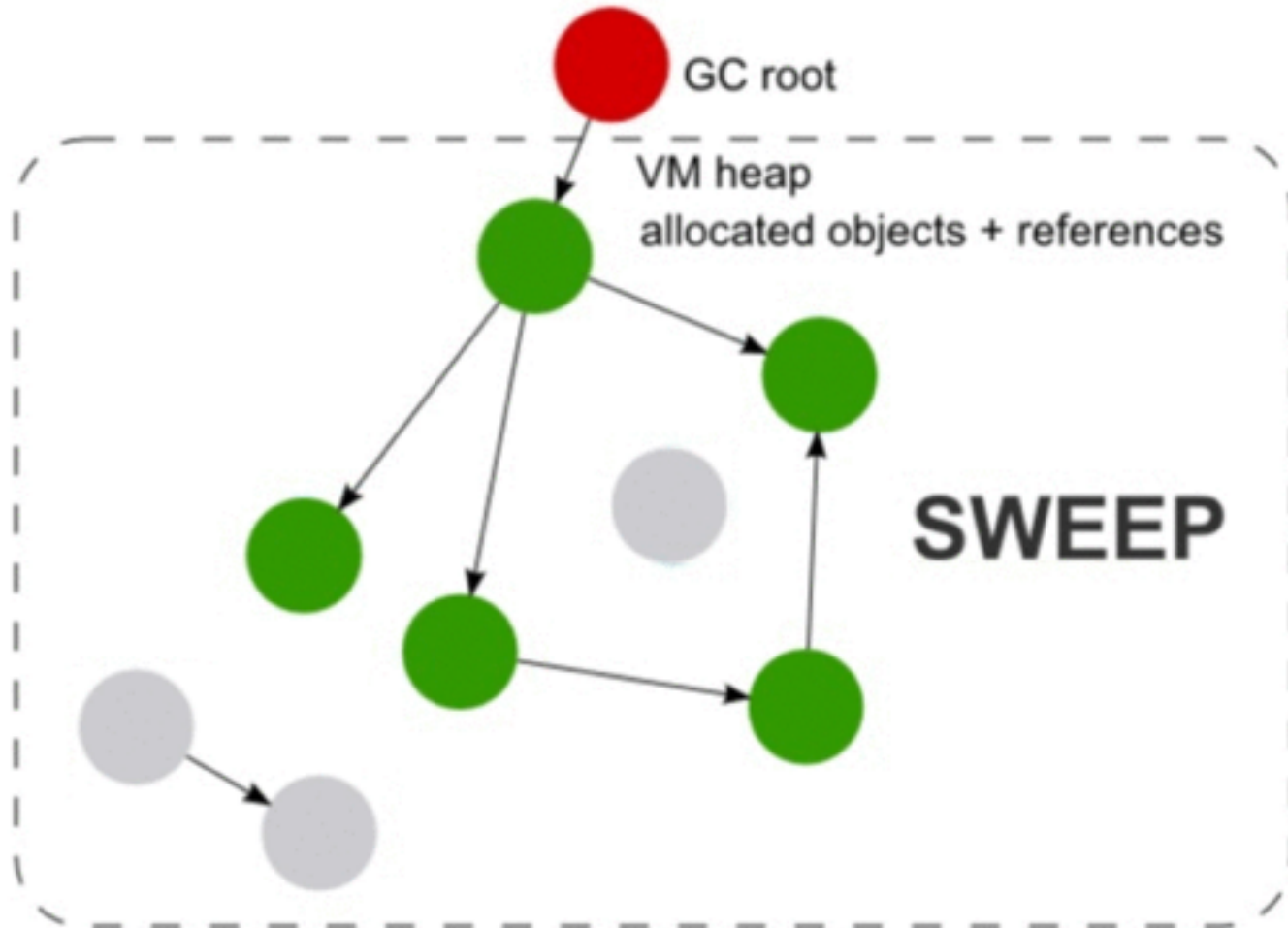
# Tracing Garbage Collection

## Mark and sweep (MARK)



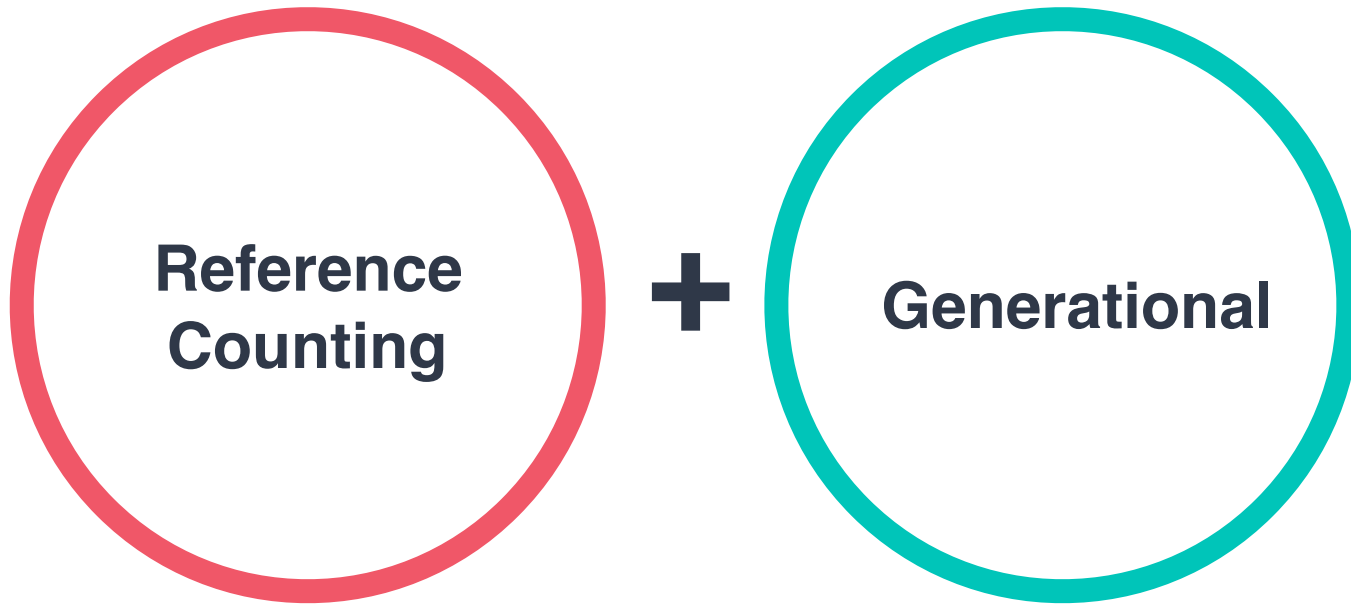
# Tracing Garbage Collection

## Mark and sweep (SWEEP)



# What does Python use?

---





**Generational Garbage Collection is based on the theory that most objects die young.**

**Python maintains a list of every object created as a program is run.**

*Actually, it makes 3.*

*generation 0*

*generation 1*

*generation 2*

*Newly created objects are stored in generation 0.*





**Only container objects with a  
refcount greater than 0 will be  
stored in a generation list.**



When the number of objects in a generation reaches a threshold, python runs a garbage collection algorithm on that generation, and any generations younger than it.

## **What happens during a generational garbage collection cycle?**

*Python makes a list for objects to discard.*

*It runs an algorithm to detect reference cycles.*

*If an object has no outside references, it's put on the discard list.*

*When the cycle is done, it frees up the objects on the discard list.*



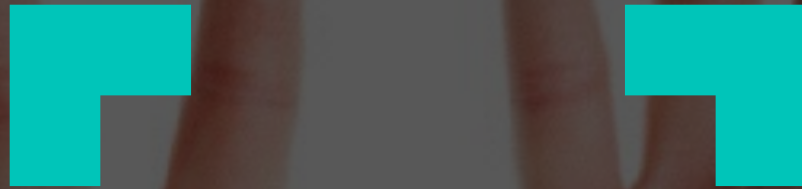
**After a garbage collection cycle, objects that survived will be promoted to the next generation.**

**Objects in the last generation (2) stay there as the program executes.**



When the ref count reaches 0, you get **immediate** clean up.

If you have a cycle, you need to **wait** for **garbage collection**.



**REFERENCE  
COUNTING  
GOTCHAS**



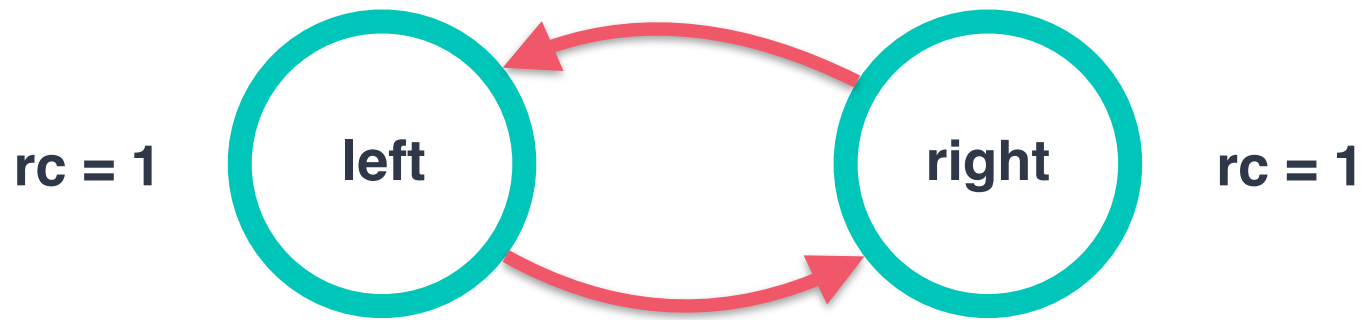


Reference counting is not generally thread-safe.

We'll see why this is a **big deal™** later.

# Remember our cycle from before?

---



Cyclical references get cleaned up by generational garbage collection.



# Cyclical Reference Cleanup

---

Except in python2 if they have a `__del__` method.

**Gotcha!**

# The `__del__` magic method

---

- Sometimes called a “destructor”
- **Not** the `del` statement.
- Runs before an object is removed from memory



slots

# What are `__slots__`?

```
class Dog(object):  
    pass
```

```
buddy = Dog()  
buddy.name = 'Buddy'
```

```
print(buddy.__dict__)
```

```
{'name': 'Buddy'}
```

# What are `__slots__`?

---

```
'Pug'.name = 'Fred'
```

```
AttributeError
```

```
Traceback (most recent call last)
```

```
----> 1 'Pug'.name = 'Fred'
```

```
AttributeError: 'str' object has no attribute  
'name'
```

# What are `__slots__`?

```
class Point(object):  
    __slots__ = ('x', 'y')
```

```
point = Point()  
point.x = 5  
point.y = 7
```

```
point.name = "Fred"
```

```
Traceback (most recent call last):  
  File "point.py", line 8, in <module>  
    point.name = "Fred"  
AttributeError: 'Point' object has no attribute  
'name'
```

What is the  
type of  
`__slots__`?



# size of dict vs. size of tuple

---

```
import sys

sys.getsizeof(dict())
sys.getsizeof(tuple())
```

**sizeof dict: 288 bytes**

**sizeof tuple: 48 bytes**

# When would we want to use `__slots__`?


---

- If we're going to be creating many instances of a class
- If we know in advance what properties the class should have



A close-up photograph of a shark swimming in clear blue water. The shark's body is light-colored with dark vertical stripes along its side. A white speech bubble with a black border is positioned over the shark's head, containing the text "WHAT'S A GIL?".

**WHAT'S A  
GIL?**

A vintage metal padlock is centered on a dark wooden surface. The padlock is made of dark metal and has a keyhole in the center. It is surrounded by four teal-colored corner brackets, two on the top and two on the bottom, forming a square frame around the text. The text is in a bold, white, sans-serif font.

**GLOBAL  
INTERPETER  
LOCK**



**Only one thread can run in the interpreter at a time.**

# Advantages / Disadvantages of a GIL

---

## Upside

Fast & Simple Garbage Collection

## Downside

In a python program, no matter how many threads exist, **only one thread will be executed at a time.**

## Want to take advantage of multiple CPUs?

---

- Use multi-processing instead of multi-threading.
- Each process will have its own GIL, it's on the developer to figure out a way to share information between processes.

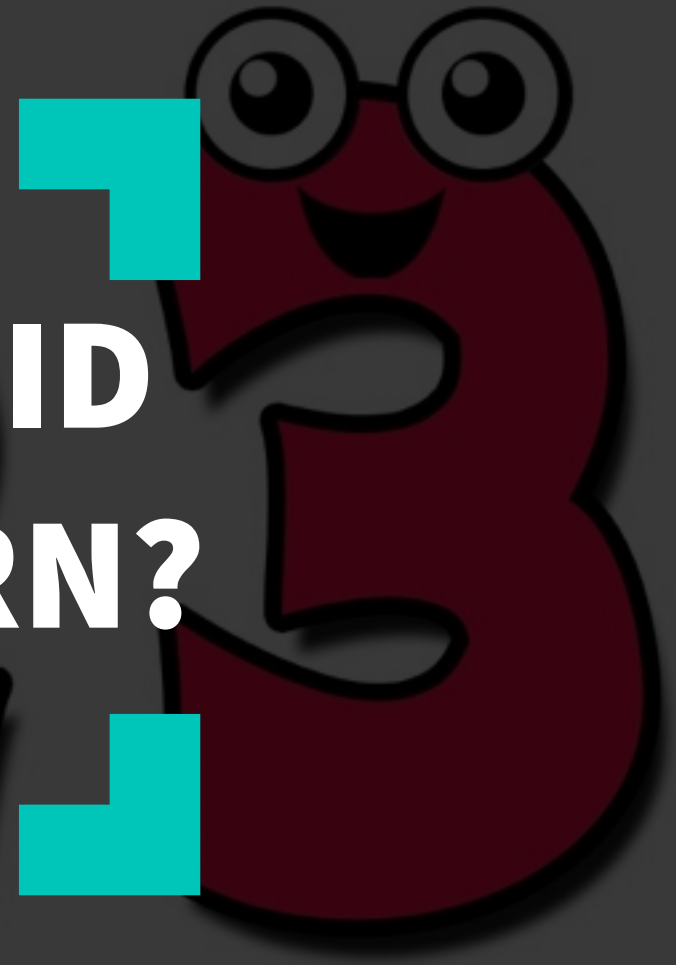


**If the GIL limits us,  
can't we just remove  
it?**



**For better or for  
worse, the GIL is  
here to stay!**





**WHAT DID  
WE LEARN?**





**Garbage collection is  
pretty good.**



**Now you know how  
memory is managed.**



**Consider  
python3**



**Or, for scientific  
applications numpy  
& pandas.**



# Thanks!



**@nnja**



**[bit.ly/memory\\_management](https://bit.ly/memory_management)**



**nina.writes.code@gmail.com**

The background of the image is a dense, overlapping field of US coins, including quarters, dimes, and nickels, in various orientations and shades of gray. Centered on this background is the text "Bonus Material" in a white, sans-serif font. The text is framed by four bright green L-shaped corner brackets: one in the top-left, one in the top-right, one in the bottom-left, and one in the bottom-right.

**Bonus  
Material**

# Additional Reading

---

- Great explanation of generational garbage collection and python's reference detection algorithm.
  - <https://www.quora.com/How-does-garbage-collection-in-Python-work>
- Weak Reference Documentation
  - <https://docs.python.org/3/library/weakref.html>
- Python Module of the Week - gc
  - <https://pymotw.com/2/gc/>
- PyPy STM - GIL less Python Interpreter
  - <http://morepypy.blogspot.com/2015/03/pypy-stm-251-released.html>
- Saving 9GB of RAM with python's `__slots__`
  - <http://tech.oyster.com/save-ram-with-python-slots/>

# Getting in-depth with the GIL

---

- Dave Beazley - Guide on how the GIL Operates
- <http://www.dabeaz.com/python/GIL.pdf>
  
- Dave Beazley - New GIL in Python 3.2
- <http://www.dabeaz.com/python/NewGIL.pdf>
  
- Dave Beazley - Inside Look at Infamous GIL Patch
- <http://dabeaz.blogspot.com/2011/08/inside-look-at-gil-removal-patch-of.html>



# Why can't we use the REPL to follow along at home?

- Because It doesn't behave like a typical python program that's being executed.
- Further reading: <http://stackoverflow.com/questions/25281892/weird-id-result-on-cpython-intobject>

## PYTHON PRE-LOADS OBJECTS

- Many objects are loaded by Python as the interpreter starts.
- Called peephole optimization.
- Numbers: -5 -> 256
- Single Letter Strings
- Common Exceptions
- Further reading: <http://akaptur.com/blog/2014/08/02/the-cpython-peephole-optimizer-and-you/>

## Common Question - Why doesn't python a python program shrink in memory after garbage collection?

---

- The freed memory is fragmented.
- i.e. It's not freed in one continuous block.
- When we say memory is freed during garbage collection, it's released back to python to use for other objects, and not necessarily to the system.
- After garbage collection, the size of the python program likely won't go down.

# How does python store container objects?

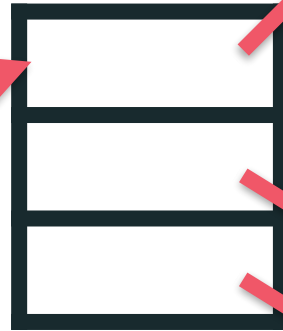
```
nums = [-10, -9, -9]
```

**nums**



## PyListObject

type	list
refcount	1
value	
size	3
capacity	10



## PyObject

Value	-10
refcount	1
type	integer

## PyObject

Value	-9
refcount	2
type	integer

# Credits

---

Big thanks to:

- Faris Chebib & The [Salt Lake City Python Meetup](#)
- The many friends & co-workers who lent me their eyes & ears, particularly [Steve Holden](#)

Special thanks to all the people who made and released these awesome resources for free:

- Presentation template by [SlidesCarnival](#)
- Photographs by [Unsplash](#)
- Icons by [iconsdb](#)

