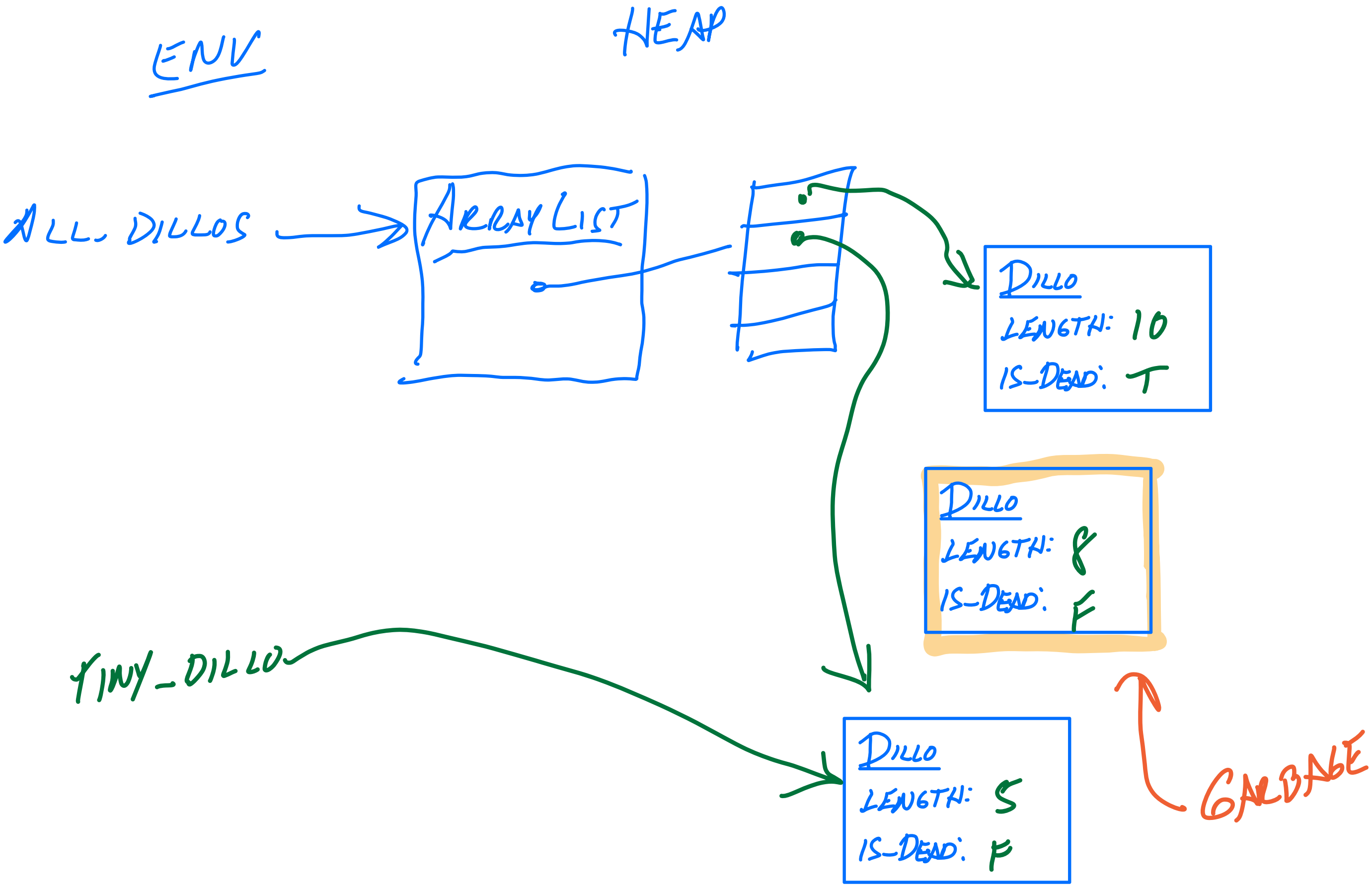
Garbage Collection

Let's think back to how objects are represented in memory...

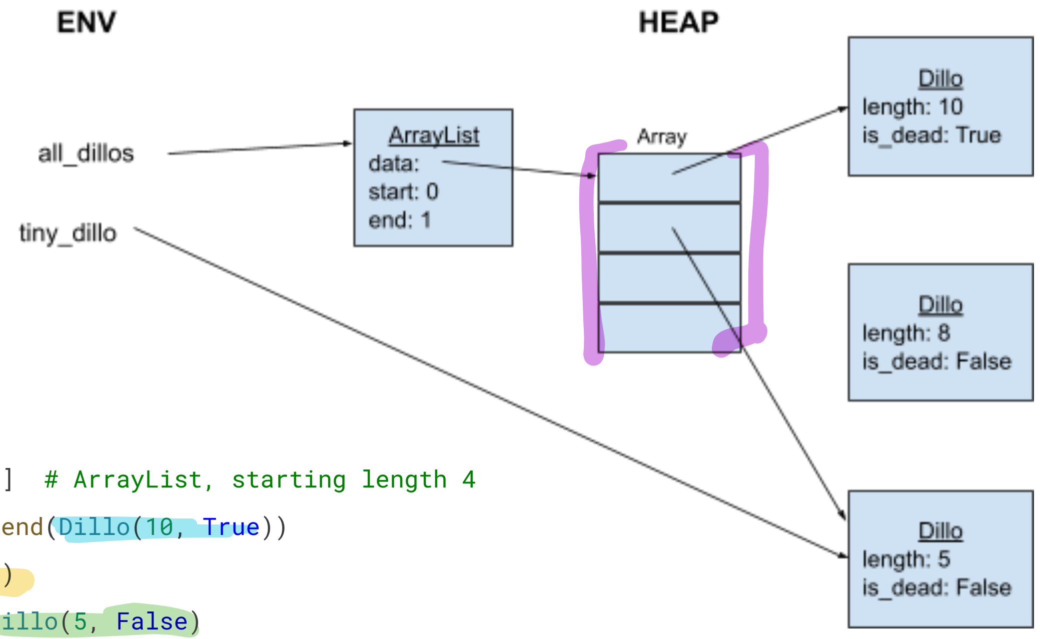
```
class Dillo:
   def __init__(self, length: int, is_dead: bool):
       self.length = length
       self.is_dead = is_dead

all_dillos = []  # ArrayList, starting length 4
all_dillos.append(Dillo(10, True))
Dillo(8, False)
tiny_dillo = Dillo(5, False)
all_dillos.append(tiny_dillo)
```



=> We can get to objects in the heap by "following" the names in the environment (and other objects they reference)
=> But here we have an object that isn't referenced by anything...

Garbage: data in the heap that the program cannot access

=> Garbage collection: process of finding grabage objects and removing them



```
all_dillos = []  # ArrayList, starting length 4
all_dillos.append(Dillo(10, True))
Dillo(8, False)
tiny_dillo = Dillo(5, False)
all_dillos.append(tiny_dillo)
```

ADDR

env		GARBAGE	heap		
all_dillos	------> @1001	NO	@	1001	ArrayList(data:@1002, start:0, end:1, size:2, cap:4)
tiny_dillo	------> @1008	NO	@	1002	@1006
		NO	@	1003	@1008
		NO	@	1004	
		NO	@	1005	
		NO	@	1006	Dillo(length: 10, is_dead: True)
		YES!	@	1007	Dillo(length: 8, is_dead: False)
		NO ✓	@	1008	Dillo(length: 5, is_dead: False)
			@	1009	free
			@	1010	free
			@	1011	free
			@	1012	free

ARRAY

SPACE RESERVED FOR ARRAY

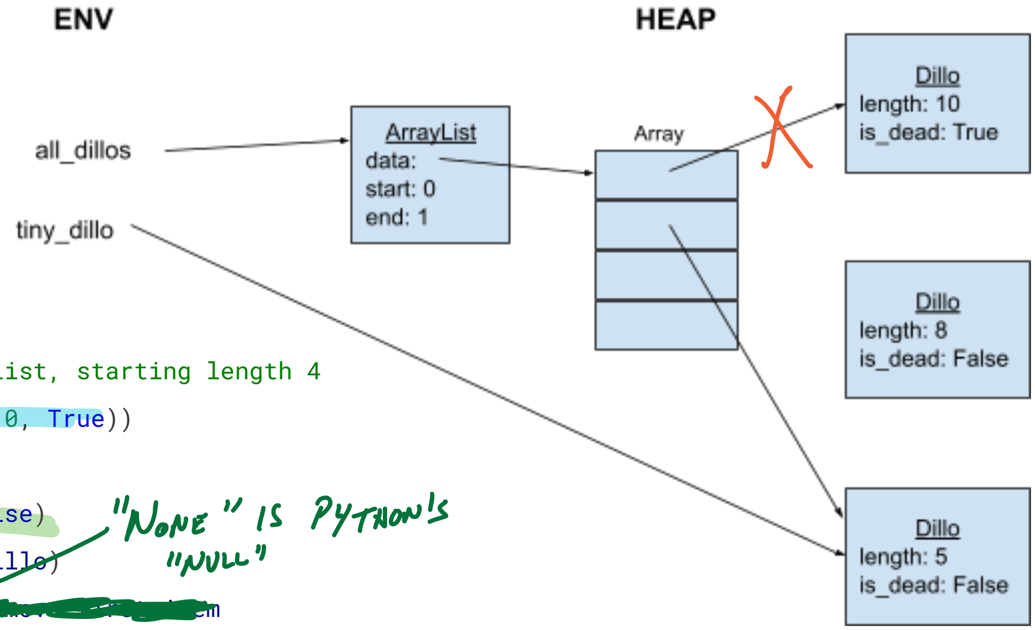
SPACES NOT BEING USED!

How to find garbage
- Follow all the names => everything we can find is by definition not garbage
- Everything else that's left is garbage

=> Of the algorithm we've seen, what is this similar to?

How it really works: GC uses DFS for each name in the environment
- **Mark each location you find as "not garbage"**
- **Anything not marked as garbage => can remove it**

What happens if we remove from the list?



```
all_dillos = []  # ArrayList, starting length 4
all_dillos.append(Dillo(10, True))
Dillo(8, False)
tiny_dillo = Dillo(5, False)
all_dillos.append(tiny_dillo)
all_dillos[0] = None # [redacted]
```

"None" is Python's "NULL"

env			heap		
all_dillos	------> @1001		@	1001	ArrayList(data:@1002, start:0, end:1, size:2, cap:4)
tiny_dillo	------> @1008		@	1002	~~@1006~~
			@	1003	@1008
			@	1004	
			@	1005	
			@	1006	Dillo(length: 10, is_dead: True) FREE
			@	1007	Dillo(length: 8, is_dead: False)
			@	1008	Dillo(length: 5, is_dead: False)
			@	1009	free
			@	1010	free
			@	1011	free
			@	1012	free

If we run the line:

all_dillos[0] = None

... we remove a reference to the Dillo with length 10. There are no other references to it, so this becomes garbage. Python (or Java)'s GC process will notice this and free up the memory, so it can be used for other things!

Extra notes on this example

1. *To remove the first element from a list in Python, it's better to write* "`all_dillos.pop(0)`". *This removes the first element, and Python will shift all other elements up, which is usually what we want (in this case, we don't care about the shifting).*

2. *What if we did* `all_dillos[1] = None` *instead? Would this create garbage? No! The Dillo with length 5 is still referenced by tiny_dillo, so it's still reachable in the environment, and therefore not garbage.*

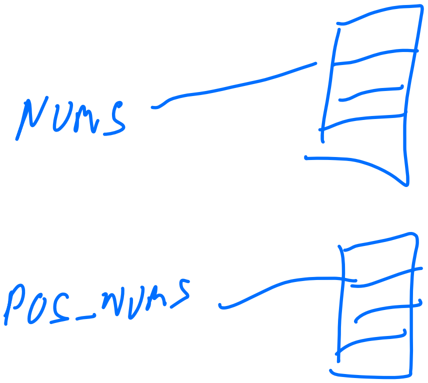
What Generates Garbage?

Example: Find the average of a list of positive numbers

What about this program? Does this create garbage?

```
# Example 1
nums = [67, 45, 0, 66, -21, 50]
pos_nums = [x for x in nums if x > 0]
avg_val = sum(pos_nums) / len(pos_nums)
print(avg_val)
```

FILTER



No. All of the objects created here (eg. the two lists) are assigned to names, so they stay in the environment.

Consider, though: do we want both of these lists in the environment? pos_nums is just a temporary variable that we used to get avg_val....

Perhaps we could design this program a bit differently so we don't keep this extra variable around? We'll discuss this more next class...

Example: what would happen if we resized the arraylist to size 8?

env					heap
all_dillos	------> @1001		@	1001	ArrayList(data:~~@1002~~ @1009, start:0, end:1, size:2, cap:4)
tiny_dillo	------> @1008		@	1002	~~@1006~~
			@	1003	~~@1008~~
			@	1004	
			@	1005	
			@	1006	Dillo(length: 10, is_dead: True)
			@	1007	Dillo(length: 8, is_dead: False)
			@	1008	Dillo(length: 5, is_dead: False)
			@	1009	~~free~~ @1006
			@	1010	~~free~~ @1008
			@	1011	~~free~~
			@	1012	~~free~~
			@	1013	~~free~~
			@	1014	~~free~~
			@	1015	~~free~~
			@	1016	~~free~~
			@	1017	free
			@	1018	free
			@	1019	free
			@	1020	free

GARBAGE!

If we need to resize the array:

- Arrays must be contiguous, so need to make a new array of size 8 in next available place in memory that has 8 slots
- References must be copied/updated to reflect new array
- Old array is no longer referenced anywhere, so it becomes garbage