Motivation
Why should I learn memory management concepts?
See behind the curtain

<table>
<thead>
<tr>
<th>struct arena_object</th>
</tr>
</thead>
<tbody>
<tr>
<td>4k  4k  4k  4k</td>
</tr>
<tr>
<td>4k  4k  4k  4k</td>
</tr>
<tr>
<td>4k  4k  4k  4k</td>
</tr>
</tbody>
</table>

Mark reachable objects, sweep rest of them

How many sweeps an objects survived?

Mark reachable objects, sweep rest of them
Learn how to Control

```python
import gc

my_list = []
my_list.append(my_list)

del my_list

gc.collect()
```

```python
import sys

name = "batuhan"
mynname = "batuhan"
ourname = "batuhan"

del name

print(sys.getrefcount("batuhan"))
```
Handle Memory Leaks

```python
1 import tracemalloc
2
3 tracemalloc.start()
4 run_some_code()
5 snapshot = tracemalloc.take_snapshot()
6 top_stats = snapshot.statistics('lineno')
7 for stat in top_stats[:10]:
8     print(stat)
```
Allocation of Memory

- Objects
- Memory Management Model
- Threshold
- Big object allocation
- Small object allocation
- Object Specifics
Everything Is An Object

- In python everything is an object

typedef struct _object {
    _PyObject_HEAD_EXTRA
    Py_ssize_t ob_refcnt;
    struct _typeobject *ob_type;
} PyObject;
Python's Memory Management Model

```
[ int ] [ dict ] [ list ] ... [ string ]     Python core |
+3 | ←------ Object-specific memory ------→ | ←-- Non-object memory →--|

[ Python's object allocator ] |
+2 | ###### Object memory ###### | ←------ Internal buffers ------→ |

[ Python's raw memory allocator (PyMem_ API) ] |
+1 | ←------ Python memory (under PyMem manager's control) ------→ |

[ Underlying general-purpose allocator (ex: C library malloc) ] |
0 | ←------ Virtual memory allocated for the python process ------→ |

| OS-specific Virtual Memory Manager (VMM) |
-1 | ←-- Kernel dynamic storage allocation & management (page-based) -----→ |

|   |   |
-2 | ←-- Physical memory: ROM/RAM -----→ | ←-- Secondary storage (swap) -----→ |
```
Small Object Threshold

---

\text{obj size} > 512 \text{ bytes} = \text{Big}

\text{obj size} < 512 \text{ bytes} = \text{Small}
Big Objects
---
- Not our concern
- Uses system allocator
Small Objects

- Managed with 3 level of abstractions
- Blocks encapsulates objects
- Pools contains same sized blocks
- Arena’s contains pool
Blocks
First level of abstractions

- 8-byte-alignment Notation
- Implementation
8-byte-alignment Notation

- The block size can be range(8, 512+1, 8).
- The size idx value can be found with (allocated space / 8) – 1

<table>
<thead>
<tr>
<th>Object Size (bytes)</th>
<th>Allocated Space (bytes)</th>
<th>Size Idx</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>9-16</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>17-24</td>
<td>24</td>
<td>2</td>
</tr>
<tr>
<td>25-32</td>
<td>32</td>
<td>3</td>
</tr>
<tr>
<td>33-40</td>
<td>40</td>
<td>4</td>
</tr>
<tr>
<td>505-512</td>
<td>512</td>
<td>63</td>
</tr>
</tbody>
</table>
Implementation

- They designed for containing python objects
- Uses 8-byte-alignment notation for better management over free blocks
- Marked as free and linked to free blocks when their object deallocated.
Pools encapsulates same sized blocks.

- Implementation
- States
Implementation

- Contains same sized blocks
- 4K Size
- Every pool has a pool_header overhead for meta information.
- Every pool linked together with nextpool & prevpool ptrs.
Implementation - Free Block

- Linked List of Blocks
- Blocks inserted whenever they freed.
States of Blocks

- **Used**
- **Full**
- **Empty**
Arenas

Encapsulates pools

- Implementation
Implementation

- Contains 64 pools.
- Size is 256kb. A big block of memory.
- System allocator only allocates space for arenas. The other abstractions uses this space.
- Also they are linked together like pools.

```c
1 struct arena_object {
2     uintptr_t address;
3     block* pool_address;
4     uint nfreepools;
5     uint ntotalpools;
6     struct pool_header* freepools
7     struct arena_object* nextarena;
8     struct arena_object* prevarena;
9   };
```
Object Specifics

- String Interning
- Small Integers
String Interning

- One object and multiple names assigned to it
- Happens in Compile Time
- By default basic strings

```python
>>> a = "batuhan"
>>> b = "batuhan"
>>> assert a is b

>>> a = "b@tuhan"
>>> b = "b@tuhan"
>>> assert a is not b

>>> a = "batuhan"
>>> b = ".join(a)
>>> assert a is not b
```
Small Integers

- between -5 ... 256
- has internal references
Garbage Collection

Deallocation of Memory

- Reference Counting
- Generational GC
What is Reference Count?

- Reference
- Ref Count

```python
1 import sys
2
3 a = 2018
4 b = [2018]
5 c = dict(a=2017, b=2018, c=2019)
6
7 sys.getrefcount(a)
```
Good Sides vs Bad Sides

- Easy to find unused object
- No need for marking
- Overhead
- No support cyclical references.
- One of the reasons of GIL
What is Generational GC?

- A GC type powered by a tracing algorithm called Mark&Sweep
- Has generations and the generations helps GC to find cyclic references.
How To Track / Manage It

- gc
- tracemalloc
import gc

def dummyf(i):
    a = []
    a.append(a)

gc.disable()
if not gc.isenabled():
    for _ in range(10):
        dummyf(_)
    collected = gc.collect()
    print(f"Total {collected} objects freed")

print(f"Track status of 'a': {gc.is_tracked('a')}")
print(f"Track status of []: {gc.is_tracked([])}")
import tracemalloc

tracemalloc.start(2)

a = dict(a='b', c='d')
b = list(range(5))

snapshot = tracemalloc.take_snapshot()

stat = snapshot.statistics('traceback')[0]
print(f"{stat.count} memory blocks: {stat.size / 1024} KiB")
for line in stat.traceback.format():
    print(line)
Debug Malloc Stats

---

```python
import sys
sys._debugmallocstats()
```

PYTHONMALLOC

- debug
- malloc_debug
- pymalloc_debug
Questions?