

# Traquer les fuites mémoires Python



Pycon 2013, Strasbourg



Victor Stinner  
victor.stinner@gmail.com

# Victor Stinner

---



- Core developer Python depuis 2010
- [github.com/haypo/](https://github.com/haypo/)
- [bitbucket.org/haypo/](https://bitbucket.org/haypo/)
- Employé par **eNovance**

# Sommaire

---

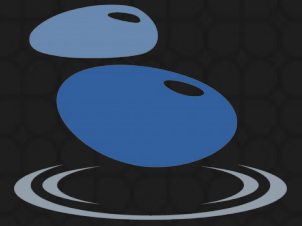


- Mémoire RSS
- Cycle référence
- Calcul manuel
- Heapy, Pympler, Melia
- PEP 445 : API malloc()
- PEP 454 : tracemalloc



# Mémoire RSS

---



- Représentatif pour le système
- Mesure grossière
- Difficile à exploiter
- Fragmentation du tas

# Fragmentation tas



2 Mo / 2 Mo

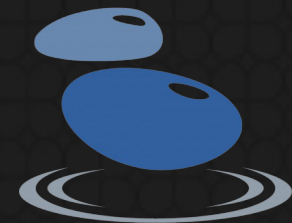


10 Mo / 10 Mo



1.5 Mo / 10 Mo

# memory\_profiler



Mem usage	Increment	Line Contents
-----------	-----------	---------------

=====

		@profile
5.97 MB	0.00 MB	def my_func():
13.61 MB	<b>7.64 MB</b>	a = [1] * (10 ** 6)
166.20 MB	<b>152.59 MB</b>	b = [2] * (10 ** 8)
13.61 MB	<b>-152.59 MB</b>	del b
13.61 MB	0.00 MB	return a

[http://pypi.python.org/pypi/memory\\_profiler](http://pypi.python.org/pypi/memory_profiler)



# Cycle référence

---



```
a.b = b  
b.a = a  
a = None  
b = None  
gc.collect()
```

# Cycle référence

---



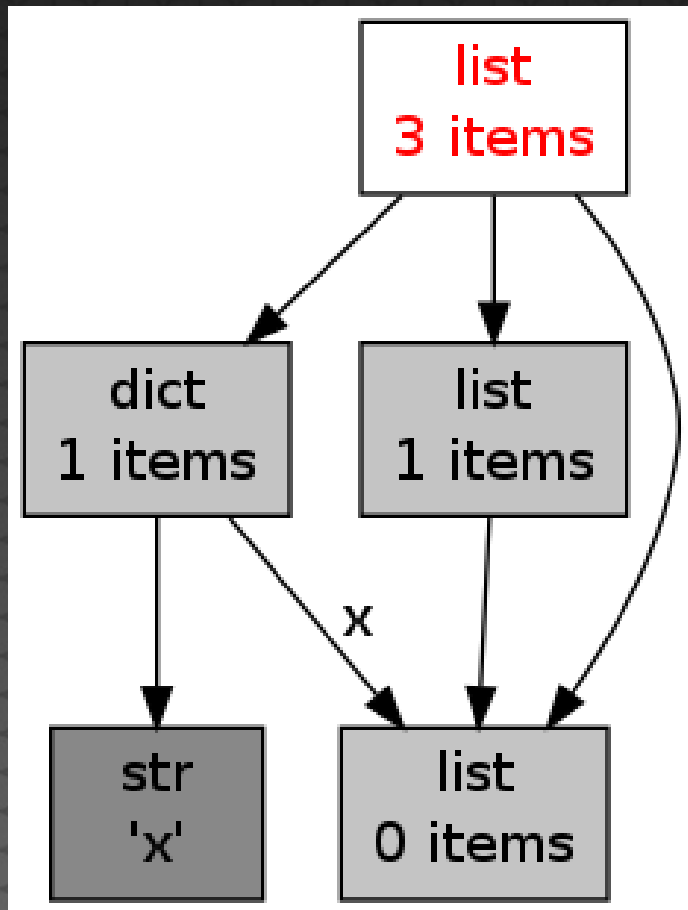
```
a.b = b
b.a = weakref.ref(a)
# a = b.a()
a = None
b = None
```



# Visualiser les liens



Project objgraph



<http://mg.pov.lt/objgraph/>

# Introspection Python



- `gc.get_objects()`
- `gc.get_referrers()`
- `gc.get_refents()`
- `id(obj)`
- `sys.getsizeof(obj)`

# Calcul manuel

---



```
>>> import gc, sys
>>> dict1 = {None: b'x' * 10000}
>>> sys.getsizeof(dict1)
296
>>> sum(sys.getsizeof(ref)
...     for ref in gc.get_referents(dict1))
10049
```



# Calcul manuel

---



```
>>> data = b'x' * 10000
>>> dict2 = {index:data
...          for index in range(500)}
>>> sum(sys.getsizeof(ref)
...      for ref in gc.get_referents(dict2))
5030496
```

# Calcul manuel

---



```
>>> refs = gc.get_referents(dict2)
>>> len(refs)
```

**1000**

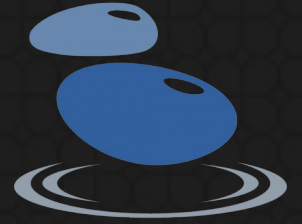
```
>>> refs = set(map(id, refs))
>>> len(refs)
```

**501**

```
>>> sum(sys.getsizeof(ref)
...     for ref in refs)
```

**16028**

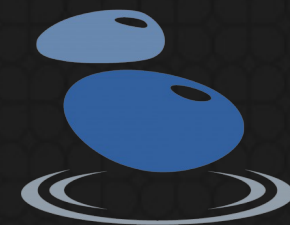
# Heapy, Pympler, Melia



- Liste l'ensemble des objets Python
- Calcul la taille des objets
- Groupe les objets par taille



# Heapy, Pympler, Melia



Total 17916 objects, 96 types,  
Total size = 1.5MiB

Count	Size	Kind
701	<b>546,460</b>	dict
<b>7,138</b>	414,639	str
208	94,016	type
1,371	93,228	code
...		

# Heapy, Pympler, Melia



- Ne trace pas toute la mémoire (ex: zlib)
- Ne donne pas l'origine des objects
- Difficile à exploiter

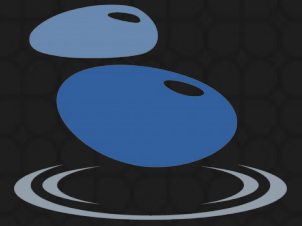
# Heapy, Pympler, Melia



- <http://guppy-pe.sourceforge.net/>
- <http://code.google.com/p/pympler/>
- <https://pypi.python.org/pypi/meliae>



# PEP 445 : API malloc()



- PyMem\_GetAllocator()
- PyMem\_SetAllocator()
- Implementé dans Python 3.4

# PEP 454 : tracemalloc

---



Memory block: 768 kB

File "test/support/\_\_init\_\_.py", line 142

File "test/support/\_\_init\_\_.py", line 206

File "test/test\_decimal.py", line 48

File "importlib/\_\_init\_\_.py", line 95

File "test/regtest.py", line 1269

# PEP 454 : tracemalloc

---



Memory block: 768 kB

File "test/support/\_\_init\_\_.py", line 142

**\_\_import\_\_(name)**

File "test/support/\_\_init\_\_.py", line 206

**\_save\_and\_remove\_module(name)**

File "test/test\_decimal.py", line 48

**C = import\_fresh\_module('decimal')**

File "importlib/\_\_init\_\_.py", line 95

**return \_bootstrap.\_gcd\_import(name)**

File "test/regtest.py", line 1269

**the\_module = importlib.import(abstest)**



# PEP 454 : tracemalloc

---



[ Top 5 ]

#1: **Lib/linecache.py:127: 225.5 kB**

#2: collections.py:368: 200.6 kB

#3: unittest/case.py:571: 108.6 kB

#4: Lib/abc.py:133: 87.3 kB

#5: Lib/shutil.py:401: 61.9 kB

10347 other: 3279.3 kB

Total allocated size: **4188.3 KB**

# PEP 454 : tracemalloc

---



[ Top 5 differences ]

- #1: test.py:4: 0.0 kB (**-5.0 kB**)
- #2: test.py:1: 2.4 kB (0.0 kB)
- #3: test.py:20: 0.6 kB (**+0.6 kB**)
- #4: test.py:8: 0.2 kB (0.0 kB)
- #5: test.py:25: 0.1 kB (0.0 kB)

# Status tracemalloc

---



- Disponible sur PyPI
- Nécessite de patcher et recompiler Python
- ... voir aussi recompiler les extensions Python écrites en C
- Patches pour Python 2.5, 2.7, 3.4



# Status PEP 454

---



- PEP 445 (API malloc) implémentée dans Python 3.4
- PEP 454 (tracemalloc) : **brouillon**, implémentation prête
- Portable ! Windows, Linux, FreeBSD, ...
- 2x plus lent et mémoire x 2
- Père : va être intégré à Python 3.4

# Suite de tracemlloc ?



- Faire accepter la PEP 454 !
- Mettre à jour les projets PyPI
- Outils pour générer des graphiques
- GUI pour exploiter les données
- GUI pour suivre l'évolution avec le temps

# Questions ?

<http://pypi.python.org/pypi/pytracemalloc>

<http://www.python.org/dev/peps/pep-0445/>

<http://www.python.org/dev/peps/pep-0454/>



Contact :

[victor.stinner@gmail.com](mailto:victor.stinner@gmail.com)



# PEP 445 (API malloc)



- Ticket ouvert en 2008
- Patch proposé en mars 2013
- Patch commité en juin 2013
- Commit reverté -> écriture PEP 445
- Meilleure API grâce à la PEP
- PEP delegate : Antoine Pitrou

# PEP 454 (tracemalloc)



- Stocke la traceback, pas juste 1 frame
- Beaucoup de code supprimé
- Code généralisé et plus haut niveau
- Échanges avec Kristján Valur Jónsson
- PEP delegate : Charles-François Natali
- Restaurant avec Charles-François

Merci David Malcom  
pour le modèle LibreOffice

<http://dmalcolm.livejournal.com/>