Zero waste:

trucchi per riciclare (la memoria) meglio

Fiscozen

Carlo Bertini
Francesco Panico

Chi siamo

Carlo Bertini

Tech team leader @Fiscozen

https://github.com/WaYdotNET

Francesco Panico

Tech team leader @Fiscozen

https://github.com/panicofr

Agenda

- ➔ Gestione della memoria in python
- ➔ Strumenti della libreria standard
- ➔ Esempi pratici

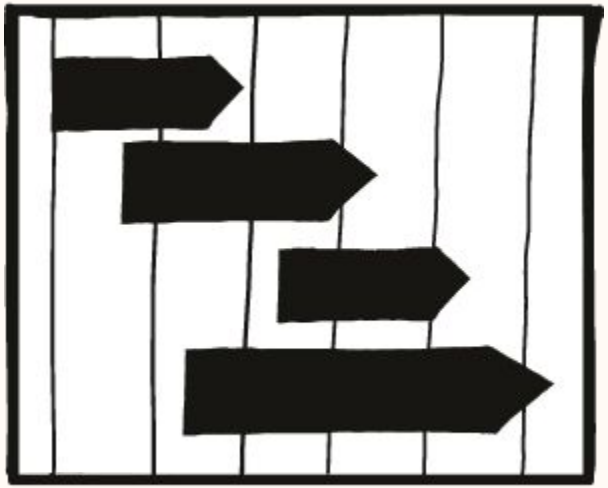
Memory management

Esempio C:

```c
int *pointer = (int*) malloc(7 * sizeof(int)); // it allocates memory
if (pointer == NULL)// it means that the memory allocation is failed
 return;

// ...

free(pointer); // pass the same pointer that was used to return malloc(). This will avoid memory leak.
pointer = NULL; // assign the pointer to NULL, otherwise it will become a dangling pointer.
```

Memory management

La gestione della memoria in Python è demandata al Memory Manager

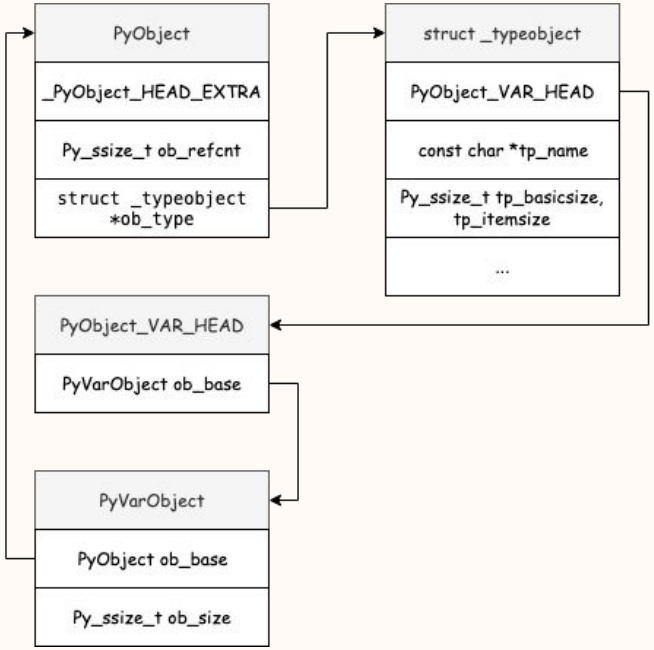


Stack e Heap

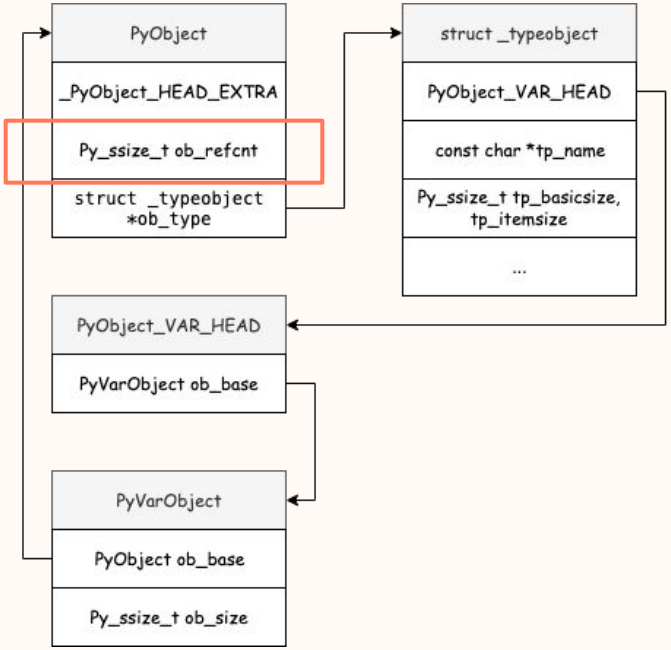
Stack

Heap

Anatomia di un oggetto CPython

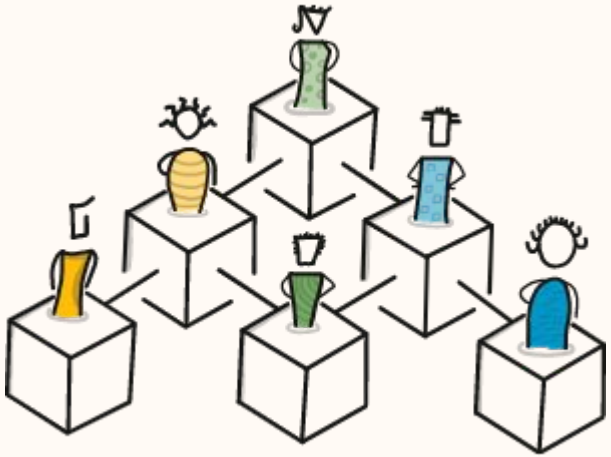


Anatomia di un oggetto CPython

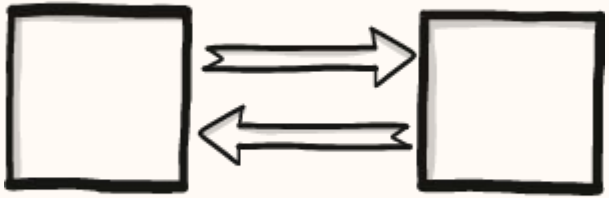


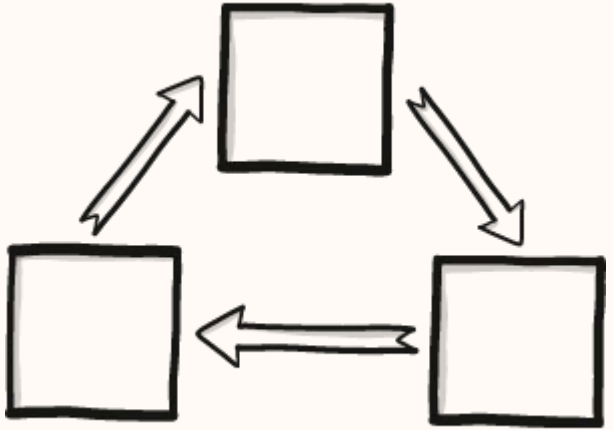
Il garbage collector

Il garbage collector: reference counting

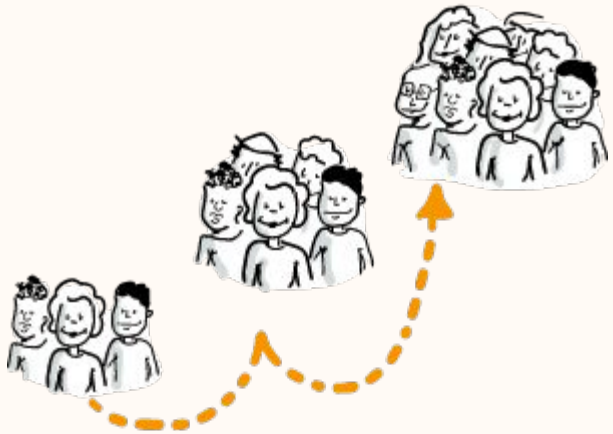


Il garbage collector: loop detection





Il garbage collector: ottimizzazioni



Memory consumption: seek and destroy

Costruire una strategia di ottimizzazione incrementale

Seek: strumenti della libreria standard

```
import sys

import gc


import tracemalloc
```

Seek: il modulo sys

```python
import sys

class MyClass:
   pass

a: MyClass | None = MyClass()

print(sys.getrefcount(a)) # 2

a2 = a
print(sys.getrefcount(a)) # 3

a2 = None
print(sys.getrefcount(a)) # 2
```

Seek: il modulo sys

```
import sys

print(f"Blocks: {sys.getallocatedblocks()}, ") # Blocks: 23308

b1 = ["AAA"]
print(f"Blocks: {sys.getallocatedblocks()}, bytes: {sys.getsizeof(b1)}, ") # Blocks: 23309,  bytes: 64

b2 = 111
print(f"Blocks: {sys.getallocatedblocks()}, bytes: {sys.getsizeof(b2)}, ") # Blocks: 23309,  bytes: 28

b2 = [1024] * 100
print(f"Blocks: {sys.getallocatedblocks()}, bytes: {sys.getsizeof(b2)}, ") # Blocks: 23310,  bytes: 856

del b2, b1
print(f"Blocks: {sys.getallocatedblocks()}, ") # Blocks: 23308
```

Seek: il modulo gc

```
import gc

existing_objects = gc.get_objects()

class MyClass:
   def __init_(self):
       self.a = 1
       self.b = "foo"

a = MyClass()
stats = {}
for o in [obj for obj in gc.get_objects() if obj not in existing_objects]:
   stats[o.__class__.__name__] = stats.get(o.__class__.__name__, 0) + 1
```

Seek: il modulo gc

```python
import gc

existing_objects = gc.get_objects()

class MyClass:
    def __init_(self):
        self.a = 1
        self.b = "foo"

a = MyClass()
stats = {}
for o in [obj for obj in gc.get_objects() if obj not in existing_objects]:
    stats[o.__class__.__name__] = stats.get(o.__class__.__name__, 0) + 1
```

```python
for type, c in stats.items():
    print(type, "\t", c)
```

```
# list                1
# function            2
# dict                1
# type                1
# tuple               1
# getset_descriptor   2
# ReferenceType       1
# MyClass             1
```

Seek: il modulo tracemalloc

Il modulo tracemalloc è uno strumento di debug per tracciare i blocchi di memoria allocati da Python.
Fornisce le seguenti informazioni:

- Restituisce il traceback di quando un oggetto viene allocato
- Calcola le statistiche dei blocchi di memoria allocati
- Calcola le differenze tra due snapshot per individuare eventuali memory leak

```
snapshot1 = tracemalloc.take_snapshot()
# porzione di codice da analizzare
snapshot2 = tracemalloc.take_snapshot()
top_stats = snapshot2.compare_to(snapshot1, "lineno")
```

Utility per il talk: measure_memory

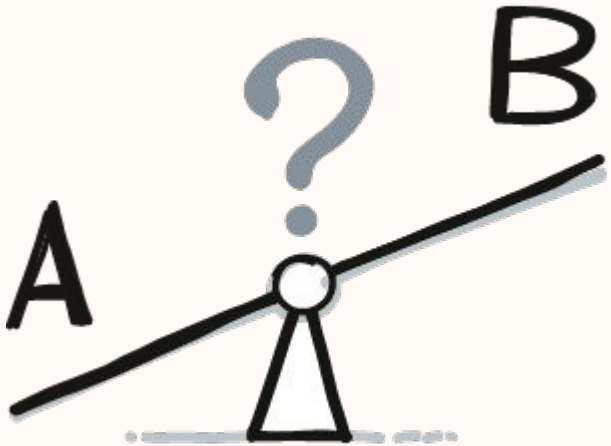
```python
@contextmanager
def measure_memory():
    tracemalloc.start()
    s_1 = tracemalloc.take_snapshot()
    base_object_count = len(gc.get_objects())
    start_time = time.time()

    yield

    s_2 = tracemalloc.take_snapshot()
    exc_time = time.time() - start_time
    top_stats = s_2.compare_to(s_1, "lineno")
```

```python
    block_count = 0
    mem_count = 0
    for stat in top_stats:
        block_count += stat.count_diff
        mem_count += stat.size_diff

    object_count = len(gc.get_objects()) - base_object_count

    print(
        f"New blocks: {block_count},",
        f"new allocated bytes: {mem_count},",
        f"objects: {object_count},",
        f"time: {exc_time}",
    )
```

Come ottimizzare il consumo di memoria

Non esiste una ricetta univoca

Come ottimizzare il consumo di memoria



Valutare il trade-off tra costo computazionale e occupazione di memoria

Destroy: keep it short!

Esempi pratici: keep it short!

```python
class A:
    def __init__(self, val: int) -> None:
        self.val: int = val
        self.double: int = compute_func(val)

    def foo(self) -> None:
        print(f"Foo: {self.double}")

    def bar(self) -> None:
        print(f"Bar: {self.double}")
```

```python
def allocator_a():
    return [A(i) for i in range(1000)]

with measure_memory():
    res_a = allocator_a()
    for i in res_a:
        i.foo()
        i.bar()
```

```
# before: New blocks: 4625, new allocated bytes: 209692, objects: 1236, time: 0.1419389247894287
```

Esempi pratici: keep it short!

```python
class A:
    def __init__(self, val: int) -> None:
        self.val: int = val

    def foo(self) -> None:
        print(f"Foo: {compute_func(self.val)}")

    def bar(self) -> None:
        print(f"Bar: {compute_func(self.val)}")
```

```python
def allocator_a():
    return [A(i) for i in range(1000)]

with measure_memory():
    res_a = allocator_a()
    for i in res_a:
        i.foo()
        i.bar()
```

```
# before: New blocks: 4625, new allocated bytes: 209692, objects: 1236, time: 0.1419389247894287
```

Esempi pratici: keep it short!

```python
class A:
    def __init__(self, val: int) -> None:
        self.val: int = val

    def foo(self) -> None:
        print(f"Foo: {compute_func(self.val)}")

    def bar(self) -> None:
        print(f"Bar: {compute_func(self.val)}")
```

```python
def allocator_a():
    return [A(i) for i in range(1000)]

with measure_memory():
    res_a = allocator_a()
    for i in res_a:
        i.foo()
        i.bar()
```

```
# before: New blocks: 4625, new allocated bytes: 209692, objects: 1236, time: 0.1419389247894287
# after:  New blocks: 3756, new allocated bytes: 185361, objects: 1030, time: 0.28345608711242676
```

Destroy: no global!

Esempi pratici: no global!

```python
# sdk.py
class Client:
    def __init__(self) -> None:
        self.log: list[str] = []

    def request(self, url: str) -> None:
        self.log.append(f"Called: {url}")


client = Client()
```

Esempi pratici: no global!

```python
from sdk import client

def foo(base_url: str) -> None:
    for r in range(100):
        client.request(base_url.format(r))
```

```python
with measure_memory():
    foo("/user/{}")
    foo("/orders/{}")
    foo("/invoices/{}")
```

```
# before: New blocks: 312, new allocated bytes: 24962, objects: 254, time: 0.0004138946533203125
```

Esempi pratici: no global!

```python
from sdk import Client

def foo(base_url: str) -> None:
    client = Client()
    for r in range(100):
        client.request(base_url.format(r))
```

```python
with measure_memory():
    foo("/user/{}")
    foo("/orders/{}")
    foo("/invoices/{}")
```

```
# before: New blocks: 312, new allocated bytes: 24962, objects: 254, time: 0.0004138946533203125
```

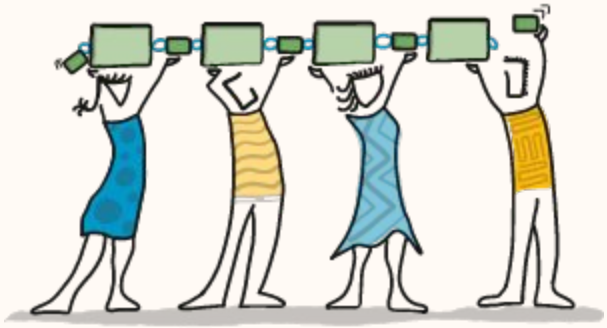
Esempi pratici: no global!

```python
from sdk import Client

def foo(base_url: str) -> None:
    client = Client()
    for r in range(100):
        client.request(base_url.format(r))
```

```python
with measure_memory():
    foo("/user/{}")
    foo("/orders/{}")
    foo("/invoices/{}")
```

```
# before: New blocks: 312, new allocated bytes: 24962, objects: 254, time: 0.0004138946533203125
# after:  New blocks: 11, new allocated bytes: 2436, objects: 88, time: 0.00034427642822265625
```

Destroy: Generator



Non allocare tutta la memoria subito, usa generatori

Esempi pratici: Generator

```
def get_result() -> list[int]:
    return [4096] * 10_000
```

```
with measure_memory():
    result = get_result()
    total = sum(result)
```

```
# before: New blocks: 12, new allocated bytes: 81984, objects: 78, time: 6.794929504394531e-05
```

Esempi pratici: Generator

```python
def get_result() -> list[int]:
    return [4096] * 10_000

def get_result():
    for _ in range(10_000):
        yield 4096
```

```python
with measure_memory():
    result = get_result()
    total = sum(result)
```

```
# before: New blocks: 12, new allocated bytes: 81984, objects: 78, time: 6.794929504394531e-05
```

Esempi pratici: Generator

```python
def get_result() -> list[int]:
    return [4096] * 10_000
```

```python
def get_result():
    for _ in range(10_000):
        yield 4096
```

```python
with measure_memory():
    result = get_result()
    total = sum(result)
```

```
# before: New blocks: 12, new allocated bytes: 81984, objects: 78, time: 6.794929504394531e-05
# after:  New blocks: 11, new allocated bytes: 2048, objects: 72, time: 0.0020530223846435547
```

Esempi pratici: Generator

```python
def get_result() -> list[int]:
    return [4096] * 10_000
```

```python
def get_result():
    for _ in range(10_000):
        yield 4096
```

```python
def get_result():
    count = 0
    while count < 10_000:
        yield 4096
        count += 1
```

```python
with measure_memory():
    result = get_result()
    total = sum(result)
```

```
# before: New blocks: 12, new allocated bytes: 81984, objects: 78, time: 6.794929504394531e-05
# after:  New blocks: 11, new allocated bytes: 2048, objects: 72, time: 0.0020530223846435547
```

Destroy: __slots__

Esempi pratici: __slots__

```python
class A:
    def __init__(self, val) -> None:
        self.val = val
```

```python
def allocator() -> list[A]:
    return [A(i) for i in range(1000)]

with measure_memory():
    res = allocator()
```

```
# before: New blocks: 3751, new allocated bytes: 184080, objects: 968, time: 0.0016489028930664062
```

Esempi pratici: __slots__

```python
class A:
    __slots__ = ["val"]

    def __init__(self, val) -> None:
        self.val = val
```

```python
def allocator() -> list[A]:
    return [A(i) for i in range(1000)]

with measure_memory():
    res = allocator()
```

```
# before: New blocks: 3751, new allocated bytes: 184080, objects: 968, time: 0.0016489028930664062
```

Esempi pratici: __slots__

```python
class A:
    __slots__ = ["val"]

    def __init__(self, val) -> None:
        self.val = val
```

```python
def allocator() -> list[A]:
    return [A(i) for i in range(1000)]

with measure_memory():
    res = allocator()
```

```
# before: New blocks: 3751, new allocated bytes: 184080, objects: 968, time: 0.0016489028930664062
# after:  New blocks: 1756, new allocated bytes: 72400, objects: 2719, time: 0.00084686279296875
```

Destroy: NamedTuple

Esempi pratici: NamedTuple

```python
class A:
    def __init__(self, val) -> None:
        self.val = val
```

```python
def allocator() -> list[A]:
    return [A(i) for i in range(1000)]

with measure_memory():
    res = allocator()
```

```
# before: New blocks: 3751, new allocated bytes: 184080, objects: 965, time: 0.001589059829711914
```

Esempi pratici: NamedTuple

```python
class A:
    def __init__(self, val) -> None:
        self.val = val


from collections import namedtuple

A = namedtuple("A", ["val"])
```

```python
def allocator() -> list[A]:
    return [A(i) for i in range(1000)]

with measure_memory():
    res = allocator()
```

```
# before: New blocks: 3751, new allocated bytes: 184080, objects: 965, time: 0.001589059829711914
```

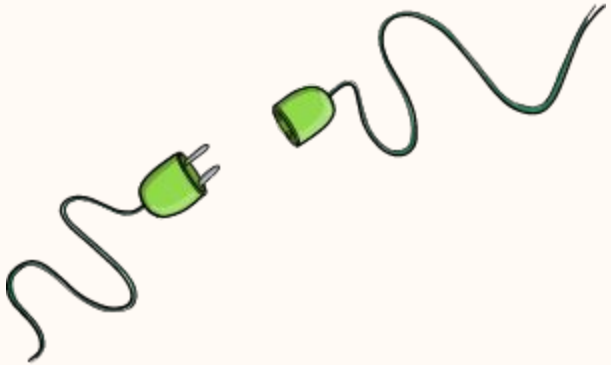
Esempi pratici: NamedTuple

```python
class A:
    def __init__(self, val) -> None:
        self.val = val


from collections import namedtuple

A = namedtuple("A", ["val"])
```

```python
def allocator() -> list[A]:
    return [A(i) for i in range(1000)]

with measure_memory():
    res = allocator()
```

```
# before: New blocks: 3751, new allocated bytes: 184080, objects: 965, time: 0.001589059829711914
# after:  New blocks: 1758, new allocated bytes: 88512, objects: 2804, time: 0.0008831024169921875
```

Destroy: Weakref



Esempi pratici: Weakref

```python
class Foo:
    def __init__(self, name: str) -> None:
        self.name = name
        self.observers = []

    def add_observers(self, *args) -> None:
        [self.observers.append(obs) for obs in args]

    def send_info(self, message: str) -> None:
        for s in self.observers:
            print(f"Info for {s.name}: {message}")
```

```python
with measure_memory():
    main = Foo("main")
    f2 =  Foo("f2")
    f3 = Foo("f3")
    f4 = Foo("f4")
    main.add_observers(f2, f3, f4)

    main.send_info("msg")
    ref_count = sys.getrefcount(f3)
    print(f"{ref_count = }")
```

```
# before: New blocks: 22, new allocated bytes: 4100, objects: 120, time: 8.0000017 ref_count = 3
```

Esempi pratici: Weakref

```python
class Foo:
    def __init__(self, name: str) -> None:
        self.name = name
        self.observers = []

    def add_observers(self, *args) -> None:
        [self.observers.append(weakref.ref(obs)) for obs in args]

    def send_info(self, message: str) -> None:
        for s in self.observers:
            if s():
                print(f"Info for {s().name}: {message}")
```

```python
with measure_memory():
    main = Foo("main")
    f2 =  Foo("f2")
    f3 = Foo("f3")
    f4 = Foo("f4")
    main.add_observers(f2, f3, f4)

    main.send_info("msg")
    ref_count = sys.getrefcount(f3)
    print(f"{ref_count = }")
```

```
# before: New blocks: 22, new allocated bytes: 4100, objects: 120, time: 8.0000017 ref_count = 3
# after:  New blocks: 22, new allocated bytes: 3780, objects: 110, time: 8.0000016 ref_count = 2
```

Conclusioni

Si può ottimizzare la memoria in python ? **SI**

Costruendo una strategia incrementale valutando il trade-off tra costo computazionale, operazioni di I/O e occupazione di memoria

Domande?

Ringraziamenti

Grazie a tutti !