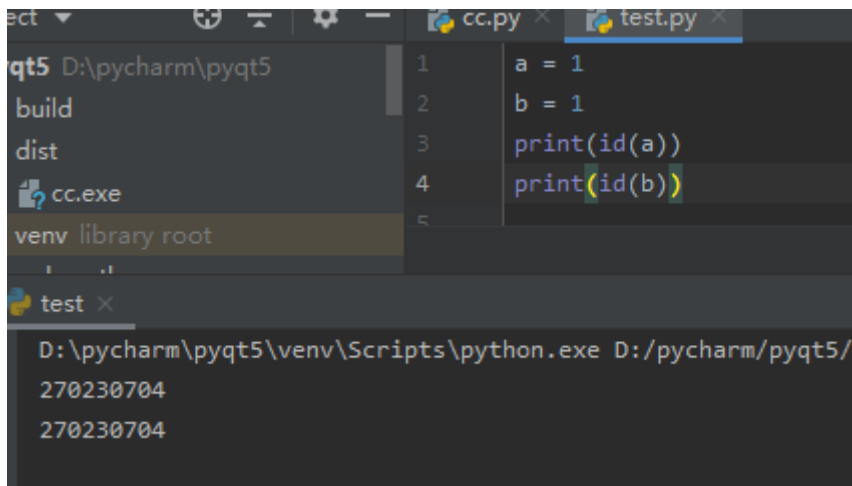


# 1.python的内存管理



The screenshot shows the PyCharm IDE with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'pyqt5' with subfolders 'build' and 'dist', and files 'cc.exe', 'venv library root', and 'test'. The code editor shows a file named 'test.py' with the following code:

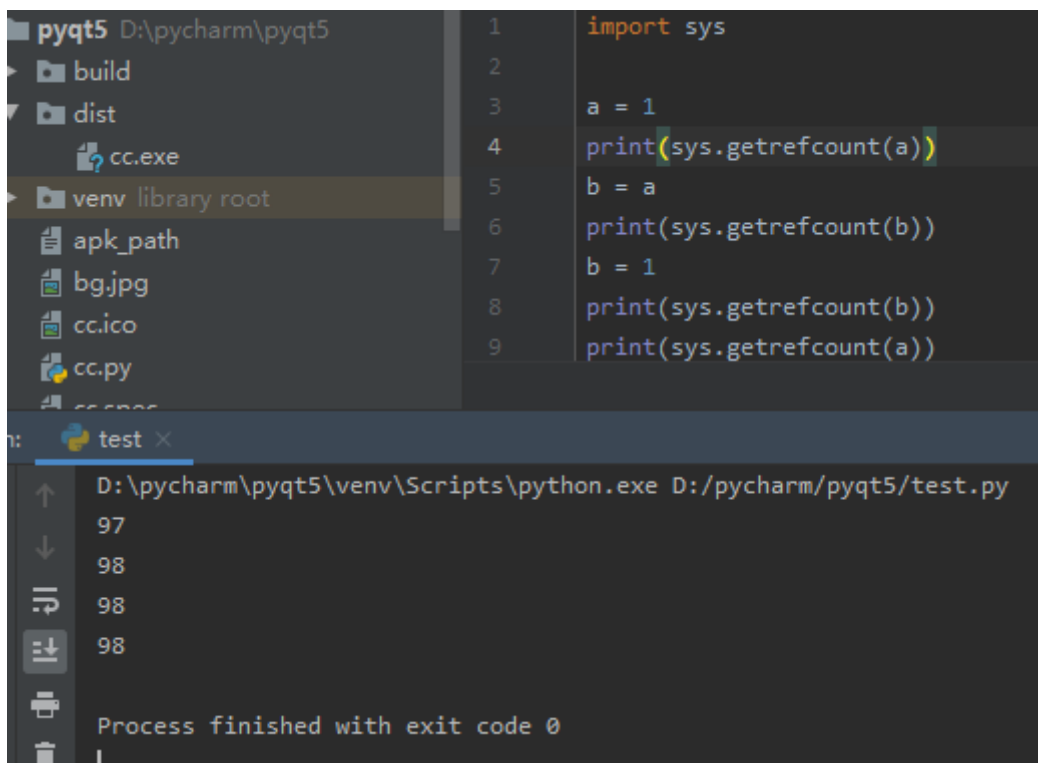
```
1 a = 1
2 b = 1
3 print(id(a))
4 print(id(b))
```

The output window at the bottom shows the command prompt with the following output:

```
D:\pycharm\pyqt5\venv\Scripts\python.exe D:/pycharm/pyqt5/
270230704
270230704
```

## 计数机制:

解释: 常量1占有固定的内存地址, 被不同的对象调用, 被调用的个数就是它的计数个数



The screenshot shows the PyCharm IDE with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'pyqt5' with subfolders 'build' and 'dist', and files 'cc.exe', 'venv library root', 'apk\_path', 'bg.jpg', 'cc.ico', 'cc.py', and 'cc.png'. The code editor shows a file named 'test.py' with the following code:

```
1 import sys
2
3 a = 1
4 print(sys.getrefcount(a))
5 b = a
6 print(sys.getrefcount(b))
7 b = 1
8 print(sys.getrefcount(b))
9 print(sys.getrefcount(a))
```

The output window at the bottom shows the command prompt with the following output:

```
D:\pycharm\pyqt5\venv\Scripts\python.exe D:/pycharm/pyqt5/test.py
97
98
98
98
Process finished with exit code 0
```

解释: sys.getrefcount去计算对象的计数个数, 不同对象调用时不会增加计数, 赋值的对象被调用时会增加计数

## 内存泄漏问题:

```
cc.py × test.py × TEST01.py ×
1 import sys
2
3 list = [1, 2]
4 a=sys.getsizeof(list)
5 print(a)
6 list.append(3)
7 print(list)
8 b=sys.getsizeof(list)
9 print(b)
10
```

pyqt5 D:\pycharm\pyqt5

build

dist

venv library root

cc.exe

apk\_path

bg.jpg

cc.ico

cc.py

cc.spec

icon.jpg

name.txt

script\_path

simulator\_path

```
4 def test():
5     list = [1, 2]
6     a = sys.getrefcount(list)
7     print(a)
8     list.append(3)
9     b = sys.getrefcount(list)
10    print(b)
11
12
13 if __name__ == "__main__":
14     test()
15
```

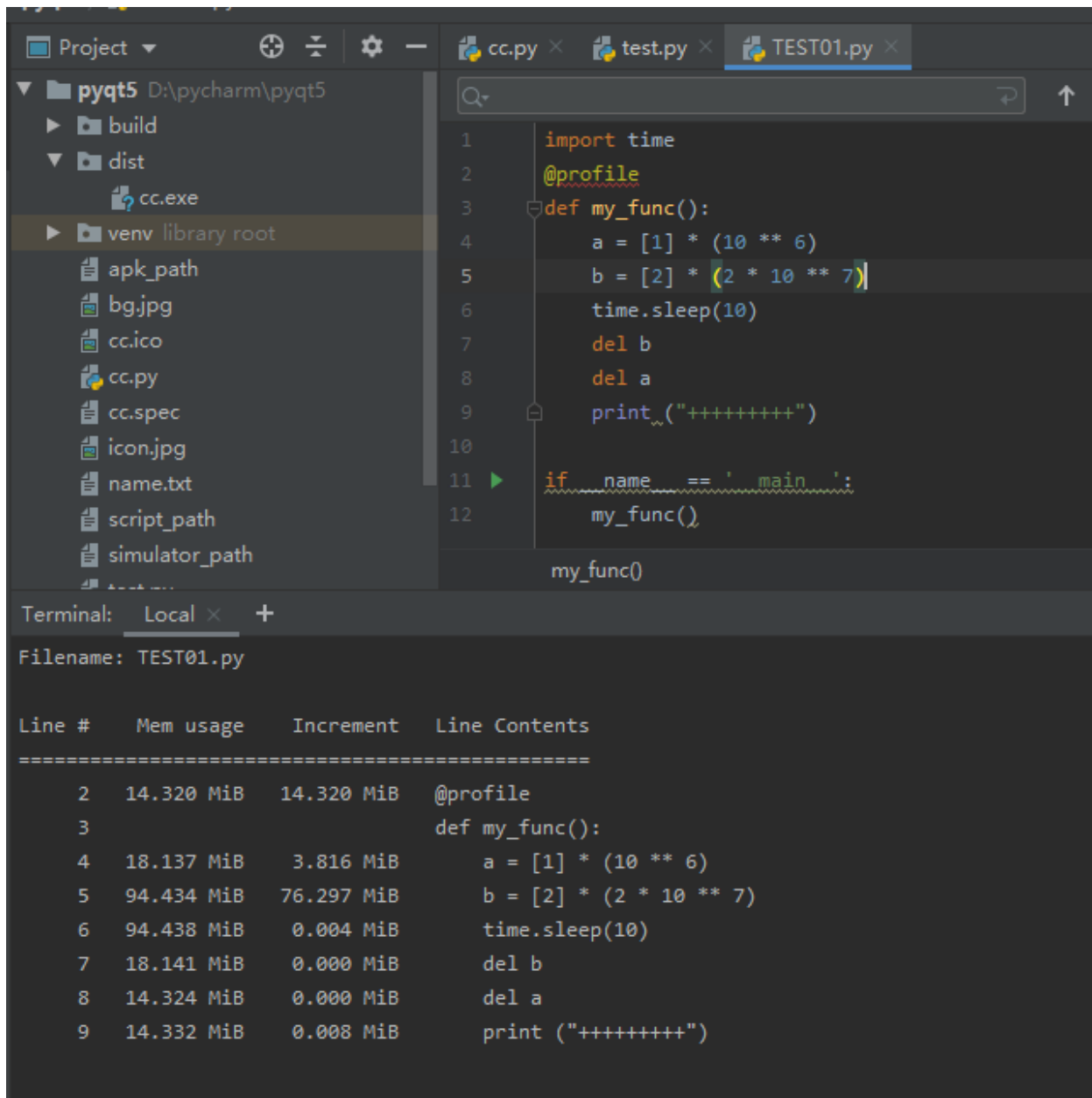
test()

Terminal: Local × +

filename: TEST01.py

| line # | Mem usage  | Increment  | Line Contents             |
|--------|------------|------------|---------------------------|
| 3      | 14.281 MiB | 14.281 MiB | @profile                  |
| 4      |            |            | def test():               |
| 5      | 14.281 MiB | 0.000 MiB  | list = [1, 2]             |
| 6      | 14.281 MiB | 0.000 MiB  | a = sys.getrefcount(list) |
| 7      | 14.289 MiB | 0.008 MiB  | print(a)                  |
| 8      | 14.289 MiB | 0.000 MiB  | list.append(3)            |
| 9      | 14.289 MiB | 0.000 MiB  | b = sys.getrefcount(list) |
| 10     | 14.289 MiB | 0.000 MiB  | print(b)                  |

解释：sys.getsizeof检测一般对象的内容占有，list容器修改不会增加内存



The screenshot shows the PyCharm IDE with a project named 'pyqt5'. The file explorer on the left shows the project structure, including a 'venv' directory. The main editor window displays a Python script named 'TEST01.py' with the following code:

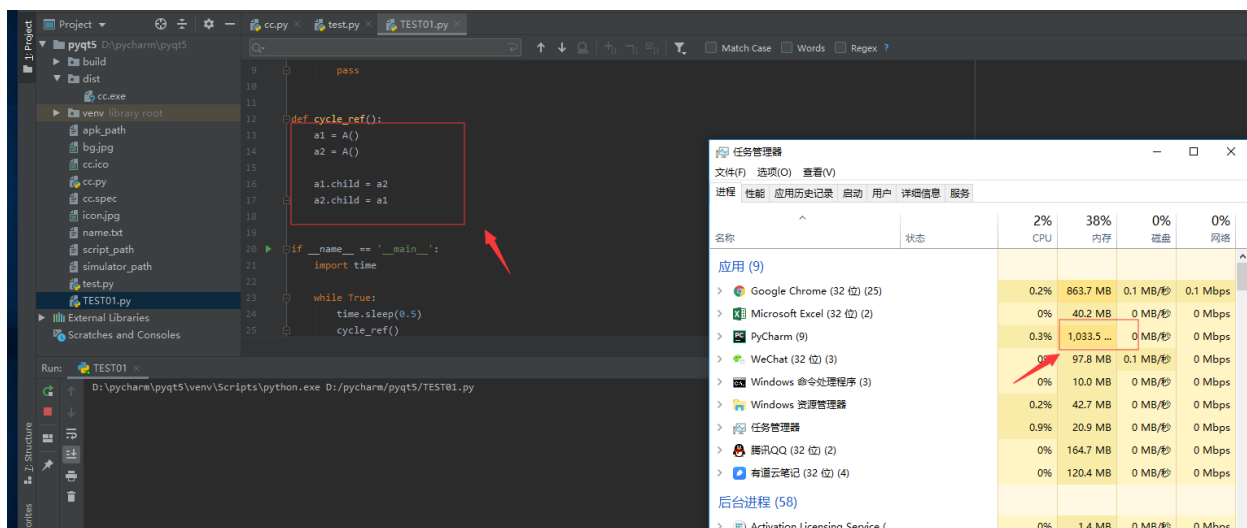
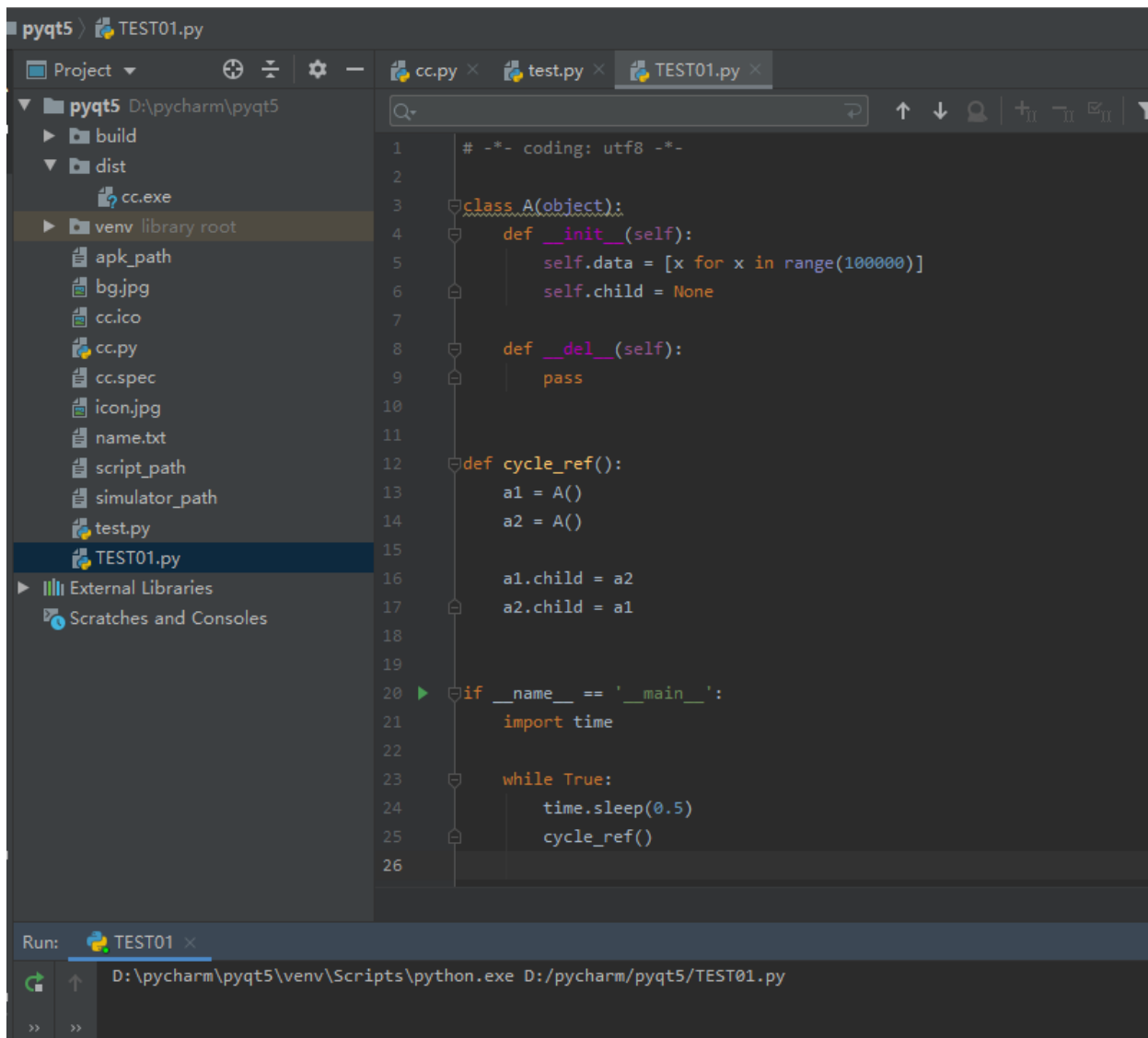
```
1 import time
2 @profile
3 def my_func():
4     a = [1] * (10 ** 6)
5     b = [2] * (2 * 10 ** 7)
6     time.sleep(10)
7     del b
8     del a
9     print("++++++")
11 if __name__ == '__main__':
12     my_func()
```

Below the editor, the terminal window shows the output of the memory\_profiler. The output is a table with columns: Line #, Mem usage, Increment, and Line Contents.

| Line # | Mem usage  | Increment  | Line Contents           |
|--------|------------|------------|-------------------------|
| 2      | 14.320 MiB | 14.320 MiB | @profile                |
| 3      |            |            | def my_func():          |
| 4      | 18.137 MiB | 3.816 MiB  | a = [1] * (10 ** 6)     |
| 5      | 94.434 MiB | 76.297 MiB | b = [2] * (2 * 10 ** 7) |
| 6      | 94.438 MiB | 0.004 MiB  | time.sleep(10)          |
| 7      | 18.141 MiB | 0.000 MiB  | del b                   |
| 8      | 14.324 MiB | 0.000 MiB  | del a                   |
| 9      | 14.332 MiB | 0.008 MiB  | print ("++++++")        |

命令:python -m memory\_profiler TEST01.py

解释: 使用memory\_profiler 去查看占用的内存变化



解释：python容器对象的，循环引用，会导致计数器无法垃圾回收，内存持续飙升

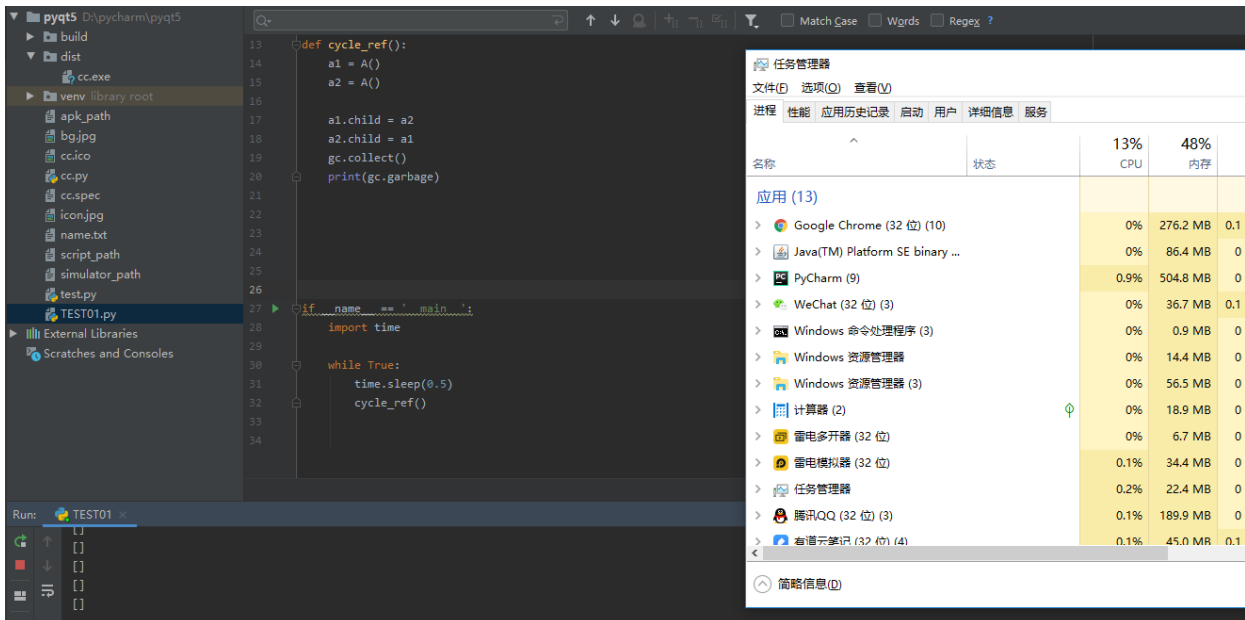
The screenshot shows the PyCharm IDE with a file named `TEST01.py` open. The code defines a class `A` with a list `data` of 100,000 elements and a `child` attribute. A `cycle_ref()` function creates two instances of `A` and sets their `child` attributes to each other, creating a circular reference. The `__del__` method is empty. The main loop calls `cycle_ref()` repeatedly. To the right, the Windows Task Manager is open, showing the 'Processes' tab. The list of applications includes Google Chrome, Microsoft Excel, PyCharm, WeChat, and Windows command processor, among others. The CPU usage is 2% and memory usage is 36%.

| 名称                                 | 状态 | 2% CPU | 36% 内存   |
|------------------------------------|----|--------|----------|
| 应用 (9)                             |    |        |          |
| Google Chrome (32 位) (25)          |    | 0%     | 794.9 MB |
| Microsoft Excel (32 位) (2)         |    | 0%     | 39.9 MB  |
| PyCharm (9)                        |    | 1.1%   | 714.0 MB |
| WeChat (32 位) (3)                  |    | 0%     | 97.6 MB  |
| Windows 命令处理程序 (3)                 |    | 0%     | 8.5 MB   |
| Windows 资源管理器                      |    | 0%     | 42.9 MB  |
| 任务管理器                              |    | 0%     | 21.6 MB  |
| 腾讯QQ (32 位) (2)                    |    | 0%     | 170.6 MB |
| 有道云笔记 (32 位) (4)                   |    | 0%     | 120.5 MB |
| 后台进程 (57)                          |    |        |          |
| Activation Licensing Service (...) |    | 0%     | 1.4 MB   |

解释：python提供的weakref来处理循环引用的问题，弱引用对象，不计入计数器，但是list、dict不支持

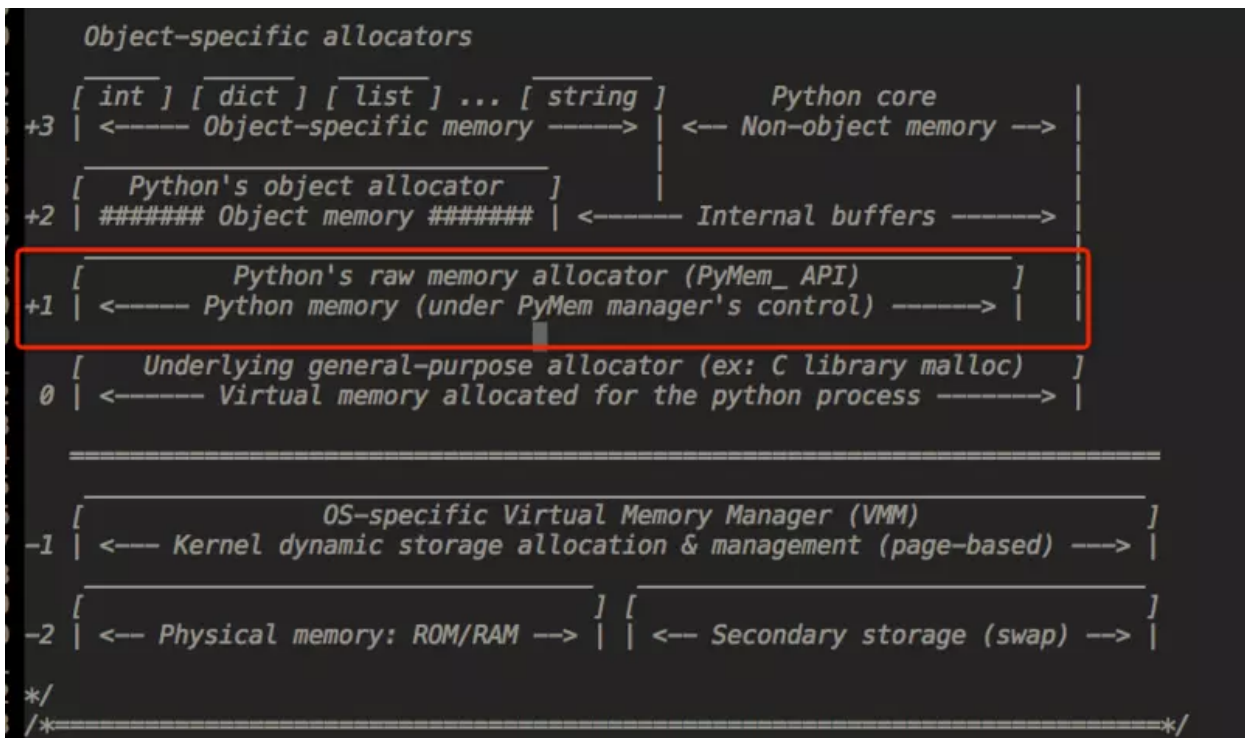
The screenshot shows the same PyCharm IDE with the `TEST01.py` file. The code is identical to the previous one, but with an additional step in the `cycle_ref()` function. A red box highlights the following code: `a1.child = None` and `a2.child = None`. This code is added to break the circular reference and prevent memory leaks. The comment above it says: `# 解除循环引用，避免内存泄露`. The main loop still calls `cycle_ref()`.

解释：写足够安全的代码，是解决内存泄漏的本质方法



解释：调用gc垃圾回收库，但是会影响性能，降低内存消耗

## 内存池机制



解释：

对int、dict、list、string对象的直接操作【不分配内存】

最小的int、dict、list、string对象分配内存【缓存对象】pymalloc函数分配<256k

最小的object对象分配内存【缓存对象】pymalloc函数分配<256k

>256kobject对象分配内存【cmalloc函数分配】