Destructor in Python

- **Destructors** are called when an object gets destroyed.
- In Python, **destructors** are not needed as much as in C++ because Python has a garbage collector that handles memory management automatically.
- The **__del__()** method is a known as a destructor method in Python.
- It is called when all references to the object have been deleted i.e when an object is garbage collected.

Syntax of destructor declaration :

```
def __del__(self):
  # body of destructor
```

Note : A reference to objects is also deleted when the object goes out of reference or when the program ends.

Example 1 : Here is the simple example of destructor. By using del keyword we deleted the all references of object 'obj', therefore destructor invoked automatically.

```
# Python program to illustrate destructor
class Employee:

    # Initializing
    def __init__(self):
        print('Employee created.')
```

```
    # Deleting (Calling destructor)
    def __del__(self):
        print('Destructor called, Employee deleted.')

obj = Employee()
del obj
```

Output

```
Employee created.
Destructor called, Employee deleted.
```

Note : The destructor was called **after the program ended** or when all the references to object are deleted i.e when the reference count becomes zero, not when object went out of scope.

Example 2: This example gives the explanation of above-mentioned note. Here, notice that the destructor is called after the 'Program End…' printed.

```
# Python program to illustrate destructor
class Employee:
    # Initializing
    def __init__(self):
        print('Employee created')
    # Calling destructor
    def __del__(self):
        print("Destructor called")
```

```
def Create_obj():
        print('Making Object...')
        obj = Employee()
        print('function end...')
        return obj
print('Calling Create_obj() function...')
obj = Create_obj()
print('Program End...')
```

Output

```
Calling Create_obj() function...
Making Object...
Employee created
function end...
Program End...
Destructor called
```

Example 3: Now, consider the following example :

```
# Python program to illustrate destructor

class A:
        def __init__(self, bb):
                self.b = bb

class B:
```

```
        def __init__(self):
                self.a = A(self)
        def __del__(self):
                print("die")


def fun():
        b = B()
```

Output

```
die
```

In this example when the function fun() is called, it creates an instance of class B which passes itself to class A, which then sets a reference to class B and resulting in a **circular reference**.

Generally, Python's garbage collector which is used to detect these types of cyclic references would remove it but in this example the use of custom destructor marks this item as "uncollectable".
Simply, it doesn't know the order in which to destroy the objects, so it leaves them.

Therefore, **if your instances are involved in circular references they will live in memory for as long as the application run.**

NOTE : The problem mentioned in example 3 is resolved in newer versions of python, but it still exists in version < 3.4 .

❖ Advantages

1) **Automatic cleanup**: Destructors provide automatic cleanup of resources used by an object when it is no longer needed. This can be especially useful in cases where resources are limited, or where failure to clean up can lead to memory leaks or other issues.

2) **Consistent behavior:** Destructors ensure that an object is properly cleaned up, regardless of how it is used or when it is destroyed. This helps to ensure consistent behavior and can help to prevent bugs and other issues.

3) **Easy to use:** Destructors are easy to implement in Python, and can be defined using the __del__() method.

4) **Supports object-oriented programming:** Destructors are an important feature of object-oriented programming, and can be used to enforce encapsulation and other principles of object-oriented design.

5) **Helps with debugging:** Destructors can be useful for debugging, as they can be used to trace the lifecycle of an object and determine when it is being destroyed.