

Python

- __init__ (Magic Method) : __init__, __del__
- __init__ (__init__)
- __del__ (__del__)

?? ???(Magic Method) : `__init__`, `__del__`

1. ?? ??

??	??
Magic Method	?? ?? ?? ?? , ?? <code>__init__</code> ?? ???? ?? ?? (?? , ?? , ? ??) ?? ???? ? ?? ??
Dunder Method	"Double Underscore" ?? ?? , ?? ?? ?? ?? ?? ??

?: `__init__` → "?? ?? (dunder init)"

2. ????? ??????

??	??	??
<code>__init__(self, ...)</code>	?? (?? ?? ? ??)	<code>p = Person("Alice")</code>
<code>__del__(self)</code>	?? (?? ?? ? ??)	<code>del p</code>
<code>__new__(cls, ...)</code>	?? ?? ?? (?? ?? ??)	?? ?? ?

- <https://dainwiki.com/books/python/page/init>
- <https://dainwiki.com/books/python/page/del>

??? ??? ?? (__init__ ??)

??? ??? ?? (init ??)

1. ??????

- `__init__` 方法是在创建对象时调用的，用于初始化对象的属性。
- `__init__` 方法通常用于设置对象的初始状态。

2. ?? ??? ??

```
class MyClass:
    def __init__(self, name):
        self.name = name
```

- `__init__()` 方法是在创建对象时调用的，用于初始化对象的属性。
- 在 `__init__` 方法中，`self` 参数是指向对象的引用，用于访问对象的属性。

3. ?? ??

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

```
p = Person("Alice", 25)
print(p.name) # Alice
print(p.age) # 25
```

- `p = Person("Alice", 25)` 调用了 `__init__()` 方法，初始化了 `name` 和 `age` 属性。

4. ??? ?? ?? ??

```
class Animal:  
    pass
```

```
a = Animal()
```

- `__init__()` 方法 初始化 对象
- 创建 对象 时 调用 方法

5. ??? ? ??

```
class Student:  
    def __init__(self, name="Unknown", grade=1):  
        self.name = name  
        self.grade = grade
```

- 创建 对象 时 调用 方法 初始化 属性

6. ??? ?????

- 创建 对象 时 调用 方法 初始化 属性
- 使用 `*args`, `**kwargs` 传递 参数

```
class Book:  
    def __init__(self, *args):  
        if len(args) == 1:  
            self.title = args[0]  
        elif len(args) == 2:  
            self.title = args[0]  
            self.author = args[1]  
        else:  
            self.title = "Unknown"
```

7. `__new__()` ?? ??

__new__	__init__
<code>__new__()</code>	객체 생성, 클래스 변수 할당
<code>__init__()</code>	객체 초기화, 인스턴스 변수 할당

8. print? ??

```
class Logger:
    def __init__(self):
        print("Logger 클래스가 생성되었습니다.")

l = Logger() # 실행: Logger 클래스가 생성되었습니다.
```

??? ??? ?? (__del__)

1. ?????

- 内存 地址 0 的 内存 地址 的 内存 地址
- 内存 (内存 地址 , 内存 地址) 的 内存

2. ?? ??

```
class MyClass:
    def __del__(self):
        print("??? ????.")
```

- 内存 地址 的 __del__ 内存 地址
- self 内存 地址 的 内存

3. ?? ??

```
class FileHandler:
    def __init__(self, filename):
        self.file = open(filename, 'w')
        print("?? ??")

    def __del__(self):
        self.file.close()
        print("?? ??")

f = FileHandler("test.txt")
del f # 内存 地址
```

4. ?? ????

- `del` 语句只能删除变量，不能删除对象
- `del` 语句删除变量时，Python 解释器会先检查该变量是否指向一个对象，如果是，则将该对象的引用计数减 1，如果为 0，则垃圾回收器 (garbage collector) 会回收该对象。如果该变量指向一个不可变对象 (如字符串、数字等)，则不会回收。
 - `del` 语句不能删除不可变对象 (如字符串、数字等)
 - `del` 语句不能删除循环引用 (circular reference)

5. `del` 语句

```
del 变量名
```

- `del` 语句只能删除变量，不能删除对象
- `del` 语句删除变量时，Python 解释器会先检查该变量是否指向一个对象，如果是，则将该对象的引用计数减 1，如果为 0，则垃圾回收器 (garbage collector) 会回收该对象。如果该变量指向一个不可变对象 (如字符串、数字等)，则不会回收。

6. `del` 语句的用法

操作	效果
<code>del 变量名</code>	删除变量名，并调用 <code>__del__</code> 方法 (如果存在)
<code>del 列表名</code>	删除列表名，并调用 <code>__del__</code> 方法 (如果存在)
<code>del 字典名</code>	删除字典名，并调用 <code>__del__</code> 方法 (如果存在)

7. `del` 语句的注意事项

- `del` 语句只能删除变量，不能删除对象
- `del` 语句删除变量时，Python 解释器会先检查该变量是否指向一个对象，如果是，则将该对象的引用计数减 1，如果为 0，则垃圾回收器 (garbage collector) 会回收该对象。如果该变量指向一个不可变对象 (如字符串、数字等)，则不会回收。

8. `with` 语句

```
with open("file.txt", "w") as f:
    f.write("Hello World")
# 文件句柄 f 在 with 语句块结束后自动关闭
```