# 对象管理

## 课程目标

* 能够说出 \_\_new\_\_ 方法的调用时机
* 能够说出 \_\_del\_\_ 方法的调用时机
* 能够使用 \_\_new\_\_ 方法实现单例模式
* 能够使用单例模式完成办公室的打印机案例

## 课程内容

* 单例模式

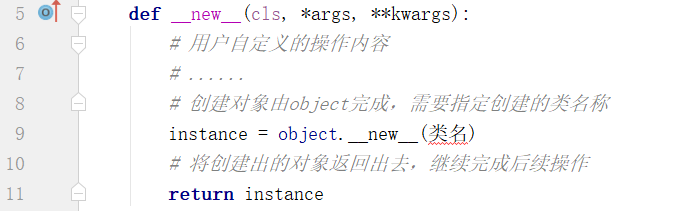
### 1.对象的创建

#### 1.1\_\_new\_\_方法作用

\_\_new\_\_方法用于定义创建对象时执行的操作，\_\_new\_\_方法仅仅为用户提供一个创建对象时干预的入口，真正执行创建对象的操作由object类中的\_\_new\_\_方法完成。

#### 1.2\_\_new\_\_方法定义

定义\_\_new\_\_方法是对object类中\_\_new\_\_方法的重写，重写该方法时必须再次调用object类中的\_\_new\_\_方法。



#### 1.3\_\_new\_\_方法调用

\_\_new\_\_方法在创建对象时自动运行，无需手工执行调用该方法。

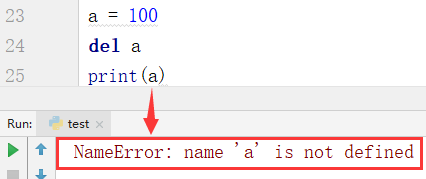
### 2.对象的删除

#### 2.1del操作作用

del操作的实质是断开变量与内存数据之间的引用，操作del执行后等同于对应变量未进行过定义。

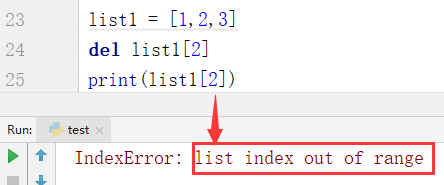
#### 2.2del操作执行范围

1.删除定义的变量：功能等同于变量未定义

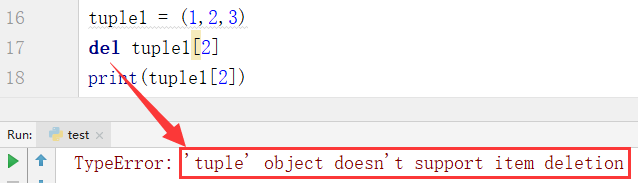


2.删除存储模型中的数据：将数据按照变量的形式看待，功能等同于未定义过对应的数据

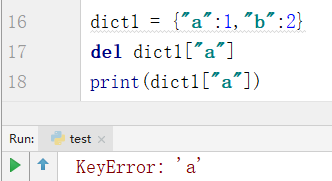
* 列表数据报错：索引不存在



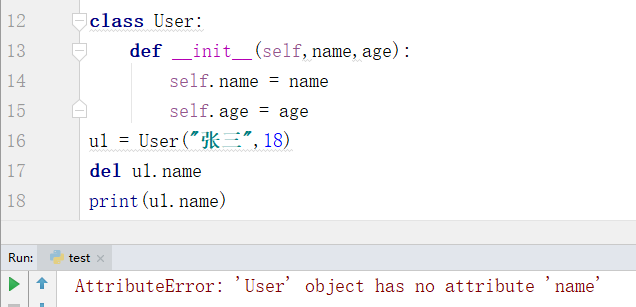
* 元组对象报错：元组对象不支持对项的删除



* 字典对象报错：键错误，键不存在



3.对象中的属性：对应对象等同于该对象未定义过该属性



#### 2.3\_\_del\_\_方法作用

\_\_del\_\_方法用于定义对象不存在任何引用时执行的操作。当对象断开了与最后一个变量之间的引用关系时，将执行该对象的\_\_new\_\_方法。

#### 2.4\_\_del\_\_方法执行范围

1.当手工执行断开变量与对象的最后一个引用时执行，即del操作

2.当程序执行完毕时，所有变量对应的引用都断开，此时执行，即退出程序

#### 2.5\_\_del\_\_方法调用

\_\_del\_\_方法在对象断开与最后一个变量之间的引用关系时自动执行，无需手工执行调用该方法。

### 3.对象判等

#### 3.1对象判等概念

当两个对象对应的引用地址相同时，称这两个对象是同一个对象。

#### 3.2对象判等语法格式

对象引用地址判等：

对象名 is 对象名 或 变量名 is 变量名

数据引用地址判定：

数据 is 数据

注：判等操作的结果是True或False

#### 3.3 ==与is的区别

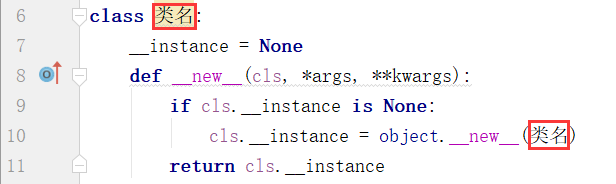
* ==操作针对判定的内容不同，规则不同。例如数值性数据比较数值的大小，不考虑是整数还是小数。字符串比较字符串的内容是否相同，不考虑引用地址是否相同。
* is操作只针对地址进行判定，只要地址相同即返回True，否则返回False

### 4.单例模式

#### 4.1单例模式的作用

控制一个类只能创建出单个实例对象。

#### 4.2单例模式书写规范



#### 4.3注意事项

1.定义的类变量要进行私有化，否则可以通过类名在类的外部访问到

2.对象的判空操作使用is进行判定

### 案例：办公室的打印机

*"""  
案例：办公室的打印机  
一台打印机可以服务于一个办公室中所有的人，完成他们的打印任务。  
分析：  
1.打印机：将要打印的任务添加到打印的任务队列中，真正的打印操作  
2.经理：将要打印的操作加入打印机中  
3.员工：将要打印的操作加入打印机中  
"""*class Manager:  
 def use\_printer(self,info,pr):  
 pr.add\_task(info)  
class Staff:  
 def use\_printer(self,info,pr):  
 pr.add\_task(info)  
class Printer:  
 \_\_instance = None  
 \_\_is\_init = False  
 def \_\_new\_\_(cls, \*args, \*\*kwargs):  
 if cls.\_\_instance is None :  
 cls.\_\_instance = object.\_\_new\_\_(Printer)  
 return cls.\_\_instance  
 def \_\_init\_\_(self):  
 if Printer.\_\_is\_init == False:  
 print("init.........")  
 self.list\_print = []  
 Printer.\_\_is\_init = True  
 def add\_task(self,info):  
 *"""添加打印信息到打印队列中"""* self.list\_print.append(info)  
 def print(self):  
 print(self.list\_print)  
  
*# 程序员A：*pr1 = Printer()  
m1 = Manager()  
m1.use\_printer("hello",pr1)  
  
*# 程序员B：*pr2 = Printer()  
st1 = Staff()  
st1.use\_printer("itcast",pr2)  
  
*# 程序员C：*pr = Printer()  
pr.print()  
  
print(id(pr1))  
print(id(pr2))  
print(id(pr))

### 案例：办公室的打印机（改造方案一）

*"""  
案例：办公室的打印机  
一台打印机可以服务于一个办公室中所有的人，完成他们的打印任务。  
分析：  
1.打印机：将要打印的任务添加到打印的任务队列中，真正的打印操作  
2.经理：将要打印的操作加入打印机中  
3.员工：将要打印的操作加入打印机中  
"""*class Manager:  
 def use\_printer(self,info,pr):  
 pr.add\_task(info)  
class Staff:  
 def use\_printer(self,info,pr):  
 pr.add\_task(info)  
class Printer:  
 \_\_instance = None  
 list\_print = []  
  
 def \_\_new\_\_(cls, \*args, \*\*kwargs):  
 if cls.\_\_instance is None :  
 cls.\_\_instance = object.\_\_new\_\_(Printer)  
 return cls.\_\_instance  
  
 def add\_task(self,info):  
 *"""添加打印信息到打印队列中"""* Printer.list\_print.append(info)  
  
 def print(self):  
 print(Printer.list\_print)  
  
*# 程序员A：*pr1 = Printer()  
m1 = Manager()  
m1.use\_printer("hello",pr1)  
  
*# 程序员B：*pr2 = Printer()  
st1 = Staff()  
st1.use\_printer("itcast",pr2)  
  
*# 程序员C：*pr = Printer()  
pr.print()

### 案例：办公室的打印机（改造方案二）

*"""  
案例：办公室的打印机  
一台打印机可以服务于一个办公室中所有的人，完成他们的打印任务。  
分析：  
1.打印机：将要打印的任务添加到打印的任务队列中，真正的打印操作  
2.经理：将要打印的操作加入打印机中  
3.员工：将要打印的操作加入打印机中  
"""*class Manager:  
 def use\_printer(self,info,pr):  
 pr.add\_task(info)  
  
class Staff:  
 def use\_printer(self,info,pr):  
 pr.add\_task(info)  
  
class Printer:  
 \_\_instance = None  
  
 def \_\_new\_\_(cls, \*args, \*\*kwargs):  
 if cls.\_\_instance is None :  
 cls.\_\_instance = object.\_\_new\_\_(Printer)  
 return cls.\_\_instance  
  
 def \_\_init\_\_(self,list\_print = []):  
 self.list\_print = list\_print  
  
 def add\_task(self,info):  
 *"""添加打印信息到打印队列中"""* self.list\_print.append(info)  
  
 def print(self):  
 print(self.list\_print)  
  
*# 程序员A：*pr1 = Printer()  
m1 = Manager()  
m1.use\_printer("hello",pr1)  
  
*# 程序员B：*pr2 = Printer()  
st1 = Staff()  
st1.use\_printer("itcast",pr2)  
  
*# 程序员B1：*pr3 = Printer()  
st2 = Staff()  
st2.use\_printer("itcast666",pr3)  
  
*# 程序员C：*pr = Printer()  
pr.print()