PYTHON

**1**请区分列表的获取切片和切片赋值：前者获取到一个新的对象（浅拷贝），而后者是对原列表的修改！ndarray相比列表，在此表现出了不同的特性：无论是获取切片还是切片赋值，ndarray都不会创建新的对象，而是共享原数据。

**2** 使用 \*n的方式初始化对象，只会得到同一个对象的多个引用；要初始化多个不同对象，请使用for循环或列表推导等操作。

**3** 列表推到和生成表达式形式上类似，前者最外层是中括号，后者是圆括号；前者生成一个列表，而后者生成一个迭代器。

**4** 相比于继承dict，继承UserDict通常是更合适的方案。原因是：继承自dict的子类，某些方法的覆写是无效的，这些方法是由非pyhton语言在底层实现的(比如get方法不会调用覆写的\_\_getitem\_\_)。而UserDict的所有方法都是在python层面实现的，因此可以被有效的覆写，只是有一个用来实现dict功能的dict成员而已。

**5** 不能在遍历字典的同时添加元素。因为添加元素可能会导致字典散列表的扩容——这会导致原来的数据的存储位置发生变化，从而使得遍历操作无法完整遍历所有数据。

**6** 默认情况下，如果两个变量或对象相等，即“变量1==变量2”，那么它们的哈希值也应该是相等的，这能防止字典的key发生不必要的混淆。这个规则对覆写提出了要求：如果覆写了某类的\_\_eq\_\_方法，那么最好也覆写它的\_\_hash\_\_方法，反之亦然，保证“变量1==变量2”和“变量1的哈希值等于变量2的哈希值”始终等价。

**7** str有个特殊的特性：s[n] == s[n:n+1]。绝大多数可切片的类都不会满足这个等式，包括list、bytearray等等。

**8** python3中函数的参数可以分为positional参数和keyword-only参数。后者在使用时，必须用“key=value”的方式传入函数。而且，无论是定义时还是使用时，keyword-only参数都必须跟在positional参数之后。如果定义函数时，需要传入keyword-only参数，而且函数没有\*args或\*\*args这样的不定长参数，那么使用如下格式分隔positional参数和keyword-only参数：def 函数名（positional参数，\*，keyword-only参数）。

**9** 调用函数时，没有设置默认值的positional参数是必须传入的。在设置默认值时，需要注意：把函数参数的默认值设为一个可变对象是一件危险的事情。因为所有函数对象会使用默认值指向的同一个对象，仅仅是多个引用而已！

1. 关于装饰器：

(1)有些代码会在引入（import）时就运行，比如装饰器中的非内嵌函数代码。

(2)使用闭包函数装饰原函数时，会修改原函数的某些属性，比如\_\_name\_\_等。在此，对闭包函数使用@functools.wraps(原函数名)，可以方便的解决该问题。

(3)为了实现装饰器，使用类比使用函数更合适，需要覆写类的\_\_call\_\_方法。

**11** global声明使函数能使用全局变量，而nonlocal声明使嵌套在内部的函数能使用外部函数定义的局部变量。

**12** 如果一个函数经常被调用，且常常需要返回相同的值，那么可以加上@functools.lru\_cache()作为优化。

**13** python使用"with"关键字管理上下文。当使用该关键字时，会调用上下文对象的\_\_enter\_\_方法；而当上下文相关的语句结束后，会调用上下文对象的\_\_exit\_\_方法。这么做的作用，本质上是简化了和上下文对象相关的try···finally···结构。把finally中需要执行的对象语句，写入对象的\_\_exit\_\_方法即可。

**14** del不删除对象，只是删除对象的引用。

**15** 当print对象时，首先查找对象是否存在\_\_str\_\_方法，如果存在则调用\_\_str\_\_，否则调用\_\_repr\_\_。由于\_\_repr\_\_方法在object中就已定义，所以它是必然存在的。

**16** 当使用“对象名.属性名=值”这样的语句为对象的属性赋值时，首先会去查找对象是否存在同名的属性，如果不存在则调用\_\_setattr\_\_，如果依然找不到，则会为对象新建一个属性并赋值，这就是Python对象的动态绑定属性。Python甚至还能动态绑定方法，虽然这不太常用。如果希望禁止对象动态绑定属性，那么可以在\_\_setattr\_\_方法中直接抛出异常，防止为对象新建属性。当获取对象的属性值时，流程也是差不多的：查找属性 （如果找不到）--> 调用\_\_getattr\_\_ (如果找不到)--> 抛出异常。

**17** 另外，使用\_\_slot\_\_也可以达到禁止对象动态绑定属性的目的，在类中添加：\_\_slot\_\_（属性1，属性2，···）。声明后，类对象便不能拥有任何非\_\_slot\_\_属性，也就无法动态绑定属性了。并且\_\_slot\_\_能优化内存的使用，减少占用的空间。但是——出于“禁止对象动态绑定属性”的目的使用\_\_slot\_\_，是不被推荐的。因为它会带来一些使用上的不便，比如它无法被继承等等。所以，除非有优化内存使用的必要，否则应谨慎使用\_\_slot\_\_。

1. python中的\*和指针可没有关系，它一般有两个作用：

（1）作为函数参数使用，意味着可将对应位置的多个参数看作为一个list，传入函数中使用（同样的\*\*和指针也没有关系，它用来将对应位置的多个参数，看作一个dic，传入函数中使用）。

(2) 在一个可迭代的变量前加"\*"，可对该变量进行自动解包（如果是字典，那么对字典的keys解包）。

**19**对于多继承来说，如果两个父类存在同名方法，默认只会调用第一个父类的方法，同样的，使用super也只会调用第一个父类的方法。如果需要使用第二个父类的同名方法或构造方法，需要指定父类名，比如：父类名.\_\_init\_\_，它调用了第二个父类的构造方法。

|  |
| --- |
|  |
|  |  |

**20** python中，可以使用“\_\_变量名”表明一个属性是私有的，并且编译器也支持这一特性。至于单下划线（\_变量名）的属性，对编译器来说它依然只是一个公共属性。但是，就如同遵守固定的命名规则一样，python程序员也应该将这些属性视为非公开的。这让单下划线的属性，具有了类似java中protected的特性。并非所有的python程序员都喜欢基于双下划线的私有属性语法规则，有些人认为使用单下划线即可——基于程序员对规则的遵守实现私有的特性，而不是由编译器强制要求。这表明了python相对java的不同之处：后者更追求严格的标准，前者更追求灵活的使用。

**21** 示例如下：

Def class A():

a = 0

def \_\_init\_\_(self, test):

self.b = test

a为类变量而b为成员变量，a可以以A.a的方式使用，而b必须通过实例对象来使用，看上去有点类似于java中静态变量和成员变量，但不同的是，python中类的实例并不会共享a（而是拷贝）！！！所以如果想要共享，那么必须通过类名来使用使用类变量。

另外，Python还具有@staticmethod和@classmethod两种方法的装饰器，带有这两种装饰器的方法，都类似于java中的静态方法，不需要实例化便可使用。不同的是，前者默认不带任何参数，而后者必须带一个cls参数（类对象本身）。后者最常见的用途是定义备选构造方法；并且在使用类成员时，避免硬编码。基于以上特性，为了代码的规范性，一般情况下：

（1）类的实例不直接使用类成员。

（2）实例的成员方法可使用类成员和实例成员；类方法仅使用类成员，静态方法不使用实例成员或类成员。