第十周问答作业

朱士杭 231300027

2024年5月2日

问题一：有关于“多态”

什么是多态：

多态，顾名思义就是“多种形态”，指的是不同的对象完成同一个相同名字的操作会形成不同的状态

python如何实现多态：

通过类继承时候的“重写”(overriding)来实现，如果子类没有重新定义与父类相同名字的方法，那么子类会根据继承父类的顺序来选择调用父类的哪个方法； 当子类与父类的类方法使用了同一个名字的时候，子类会优先调用自己重新定义的类方法
举例说明python实现多态例子如下：

```
#举例说明python实现多态
class Creature:
    def __init__(self):
        self.lifelong=100
        self.name="鲸鱼"
        self.food="鱼虾"
    def eat_food(self):
        print(f"{self.name}吃{self.food}")
class Human(Creature):
    def __init__(self):
        super().__init__()
        self.name="人类"
        self.food="什么都吃"
    def eat_food(self):
        print(self.food)
        print("此处发生了重写overriding实现了多态")
```

问题二：类共有的几个属性

注意这里要举例的是类共有的属性而不是类共有的方法，因此像__init__等方法这里并不会涉及

属性：

__name__属性：返回类的名称
__annotations__属性：返回一个字典其中包含类中的类型注解，其中键为变量/函数名，值为其类型

__basicsize__属性：返回这个类所占用基本内存空间（即在未初始化之前）的大小
__bases__属性：返回这个类所继承的所有基类的名字及其类型
__dict__属性：返回一个字典，其中包含module、__init__函数等用的是哪些函数
__module__属性：返回当前类所属的模块，在import模块的时候可以查找器属于哪个模块下的类
__doc__属性：返回的是类下面的注释文档"""里面的内容

为什么所有类需要这些共同的属性

这难道不是显而易见嘛，所有的类肯定需要共同的属性比如类的名字是什么、类基本大小是多少、类所继承了哪些基类、类下面有哪些基本的函数、类属于哪个模块py文件下等等一系列的属性

问题三：循环导入模块

在这个时候调用package模块时是不正常的
这个时候会产生循环导入的问题，当导入class1.py模块时，初始化的时候会执行from class2 import class2的语句， 此时又会导入class2.py模块，此时又要将class2导入进来，又会执行from class1 import class1的语句又转而执行导入class1模块， 里面又有导入class2的语句，从而形成了无限循环递归，最终一定会导致数据溢出。
这就好比调用函数的时候，函数1中调用了函数2，函数2中又调用了函数1，形成了死循环。这也好比在递归调用自身的时候没有返回条件导致无限递归下去

问题四：calculate模块

创建一个calculate.py文件，在该文件下代码如下：

```
def add(a,b):
    '''返回两数之和'''
    return a+b
def subtract(a,b):
    '''返回两数之差'''
    return a-b
def multiply(a,b):
    '''返回两数之积'''
    return a*b
def divide(a,b):
    '''返回两数之商'''
    return a/b
if __name__=="__main__":
    print("3+4的结果为：",add(3,4))
    print("3-4的结果为：",subtract(3,4))
    print("3*4的结果为：",multiply(3,4))
    print("3/4的结果为：",divide(3,4))
```

在主脚本main.py文件下代码结果如下：

```
import calculate as cl
print("3+4的结果为：",cl.add(3,4))
print("3-4的结果为：",cl.subtract(3,4))
print("3*4的结果为：",cl.multiply(3,4))
```

```
print("3/4的结果为：",cl.divide(3,4))
```

问题五：try中加入return语句finally是否还执行

在原先那个多态py文件中加入以下代码来测试：

```
def set_person():
    try:
        person1=Human()
        person1.name="朱士杭"
        return
    except Exception:
        pass
    else:
        pass
    finally:
        print("当在try语句中return之后finally语句仍旧会执行")
        pass
set_person()
```

最后发现会输出字符串，表明在try语句当中加入return语句对于finally语句执行是不影响的。
其实在理论上去解释这个点感觉也挺好理解的，因为finally语句无论是否会出现异常都会执行， 如果没有出现异常那么最后try语句也会返回到一开始执行的地方。
当程序执行到return语句时也会执行finally中的一些垃圾回收处理代码，比如手动del掉对象等等操作

问题六：使用上下文管理器发生异常

用with语句使用上下文管理器的时候，如果这个时候发生异常，解释器会自动抛出异常并被except语句接收（虽然在with里面不用写但是解释器会帮你执行）， 然后执行完异常处理操作之后，上下文管理器会自动帮你执行一些“售后服务”，比如说清理掉一些没有用的对象清空内存、垃圾回收机制、保存并关闭文件等操作。

问题七：相对导入与绝对导入

区别

绝对导入与相对导入就好比绝对路径与相对路径的关系，前者需要将包内路径写的明明白白的，后者只需要写明从哪里导入模块即可
假设我们现在有这样的一个包结构：

```
package/
    __init__.py
    subpackage1/
        __init__.py
        moduleX.py
        moduleY.py
    subpackage2/
```

```
        __init__.py
        moduleZ.py
    moduleA.py
    moduleB.py
```

那么假设现在在moduleA.py文件里面导入其他模块
如果要导入moduleX模块，则绝对导入方法（默认）为： import package.subpackage1.moduleX或者 from package.subpackage1 import moduleX
如果采用相对导入的方法的话则为：from .subpackage1 import moduleX
在这里一个.表示在当前层级中去寻找subpackage1，如果是两个.的话表示在上一个层级去寻找

使用场景

绝对导入：显式导入，对于层级不深的包内部模块互相导入以及外部文件导入的时候会比较简单明了不容易犯错
相对导入：隐式导入，适用于那些层级很深的包，每次调用的时候都要写一大堆路径很麻烦，直接用相对导入反而简单而且对于就在同一个文件夹下面的模块之间的导入时优势更为明显

优劣势

绝对导入：优势是不容易出错在本地部署的时候只要位置不发生移动基本不会出现问题
劣势是包层次深了以后显得冗杂以及对路径修改之后所有其他的路径都要发生修改，很麻烦
相对导入：优势是在同一级或者在同上一级下时很方便
但劣势就是在两个离得很远的分支下调用会及其困难，要写好多好多.点，太麻烦