1. **详细解释一下什么是元类**
2. 类和对象  
    1.什么是类：类是面向对象的编程语言实现信息封装的基础概念，类是一种用户定 义的类型，通俗点，一种封装若干属性和方法的结构就叫做类，类的实例化，就是对象， 所以也可以说，类就是用来创建对象的代码片段。  
    2.什么是对象：对象是现实中的物体或实例。对象是由类实例化而来的，对象是由 类创建的，一切万物皆对象。这句话对吗？讲完这个知识点，我们再回来求证这句话到 底是不是彻底的对。  
    3.类的特殊特性：类，本身，也是一个对象！  
    4.怎么理解：其实，在你定义一个类的时候，就会在内存中创建一个名为类名的对 象  
    5.对象的特征：  
    1.将他存储在一个变量里  
    2.复制它  
    3.给他添加属性  
    4.当做参数传递给函数   
    经过如上概念，对类，应该有了最最基本的理解。但是，问题继续来了：对象不可能凭空产生，它一定是由某些'事物'生成的。既然类也是一个对象，那不意外，类，同样遵循这个规律  
    当你在使用class关键字的时候，Python就会自动创建这个'对象'。但是，虽为自动创建，但是，同样可以手动实现

2.type函数

很常用，对吧？返回一个对象的类型。ok，我们现在就玩玩这个type  
 我们在命令行下随便调试，如下：  
 anwc@anweichao-pc:~$ ipython3  
 Python 3.6.5 (default, Apr 1 2018, 05:46:30)  
 Type **'copyright'**, **'credits' or 'license' for** more information  
 IPython 6.5.0 -- An enhanced Interactive Python. Type **'?' for** help.  
  
 In [1]: type(1)  
 Out[1]: int  
  
 In [2]: type(int)  
 Out[2]: type  
  
 In [3]: type(**'1'**)  
 Out[3]: str  
  
 In [4]: type(str)  
 Out[4]: type  
  
 In [5]: type(1.11)  
 Out[5]: float  
  
 In [6]: type(float)  
 Out[6]: type  
  
 In [7]: type(**True**)  
 Out[7]: bool  
  
 In [8]: type(bool)  
 Out[8]: type  
  
 In [9]: type([1,2,3])  
 Out[9]: list  
  
 In [10]: type(list)  
 Out[10]: type  
  
 In [11]: type((1,2,3))  
 Out[11]: tuple  
  
 In [12]: type(tuple)  
 Out[12]: type  
  
 In [13]: type({**'1'**:2})  
 Out[13]: dict  
  
 In [14]: type(dict)  
 Out[14]: type  
  
 以上是type最常用的方式，没错，只传递一个参数，作用很简单，返回对象的类型。而此时无论你是type(int)，还是type(dict),亦或是type(tuple)，返回的统统都是type！\_ok，聪明的同学已经恍然大悟，这个被返回的type，就是Python中的元类！  
 1.type函数传递一个参数的用法：当只传递一个参数的时候，其功能单一，仅仅知识返回对象的类型  
 2.type函数传递三个参数的时候：type是类最根本的元类，此时，用来创建类！  
 此时type函数语法：  
 类名 = type(String,tuple,dict)  
 参数解释：  
 String：字符型参数，类的名称  
 tuple:元组，继承关系  
 dict：字典，定义属性及方法  
 继续，我们在命令行中看看我们如何手动创建类：  
 anwc@anweichao-pc:~$ ipython3  
 Python 3.6.5 (default, Apr 1 2018, 05:46:30)  
 Type **'copyright'**, **'credits' or 'license' for** more information  
 IPython 6.5.0 -- An enhanced Interactive Python. Type **'?' for** help.  
  
 In [1]: Myancestor = type(**'Myancestor'**,(),{})  
  
 In [2]: Myancestor  
 Out[2]: \_\_main\_\_.Myancestor  
  
 In [3]: print(Myancestor)  
 <**class '\_\_main\_\_.Myancestor'**>  
  
 此时你手动创建了一个类Myancestor，我们来看看它的元类是谁：  
  
 In [4]: type(Myancestor)  
 Out[4]: type  
  
 看到了吗？type函数手动创建的类，其元类仍是type！  
 继续玩，我们手动创建类的时候还能给其设定继承关系和成员属性及方法：  
 In [5]: MyOld = type(**'MyOld'**,(Myancestor,),{**'name'**:**'子代'**})  
  
 In [6]: MyOld.\_\_mro\_\_  
 Out[6]: (\_\_main\_\_.MyOld, \_\_main\_\_.Myancestor, object)  
  
 添加方法？很简单，先定义好一个函数然后传进字典中即可：  
 In [7]: **def** Myfun():  
 ...: **pass** ...:  
 ...:  
  
 In [8]:  
  
 In [8]: MyBaba = type(**'MyBaba'**,(MyOld,),{**'name'**:**'子代1'**,**'Myfun'**:Myfun})  
  
 3.元类：  
 其实很好理解，创建类的类...有点拗口...Python中，类也是对象，这样的对象就 是通过类来创建的。元类就是'类的类'  
 刚才我们玩坏了的type，就是一个元类，Python中所有的类都是由type创建的！  
 继续追问：type既然是元类，那type是对象吗？type的类型是什么？ok，'Python 中一切万物皆对象'这句话，因为type元类，止步于此。这句话，我们重新概括：Python 中，除了type,一切万物皆对象！  
 4.结语：

元类，是一种99%的人都不需要关心的深度魔法。当你好奇你是否需要懂它，了解它时，恰恰说明你并不需要它，  
 而明确需要它的人不需要找任何原因解释为什么需要。元类，是你注定得不到搞不明白的东西！

1. **我如何修改某一个类的元类？**

Class 类名(metaclass=元类名称)

Or

Class 类名([继承关系]):

\_\_metaclass = 元类名称

1. **@staticmethod和@classmethod**

类方法

    定义：使用装饰器@classmethod。第一个参数必须是当前类对象，该参数名一般约定为“cls”，通过它来传递类的属性和方法（不能传实例的属性和方法）；

    调用：实例对象和类对象都可以调用。

静态方法

    定义：使用装饰器@staticmethod。参数随意，没有“self”和“cls”参数，但是方法体中不能使用类或实例的任何属性和方法；

    调用：实例对象和类对象都可以调用。

1. **类变量，实例变量，谁可以访问，谁不可以访问？**

Python变量的本质：被赋值  
 1 普通python变量(非类相关变量)很容易理解，在被赋值后即变量存在，可读可写  
 2 Python类的变量（类变量和实例对象变量）以某种方式在某地方被赋值，即存在，可被读写  
Python类变量被赋值  
     （1）类的设计里，  
             class里def外，通过变量名能被赋值  
             def里通过类对象即类名字的点运算变量名可被赋值  
      （2）程序里  
             通过类对象(类名字)的点运算类名字也可被赋值  
Python实例对象变量被赋值  
    （1）类的设计时  
            def里通过self点运算变量名能被赋值，不一定非在init里，其他已被调用的方法函数里也行  
    （2）程序里  
            通过实例对象的点运算变量名可被赋值

1. **静态方法，类方法，实例方法，谁可以调用，谁不可以调用？**

python面向对象中各种方法的具体使用情况  
**class** A(object):  
 **def** \_\_init\_\_(self):  
 print(**"A类的构造方法被调用啦！"**)  
  
 **def** \_\_del\_\_(self):  
 print(**"A类的析构方法被调用啦！"**)  
 *#实例方法* **def** fun(self):  
 print(**"A类的实例方法被调用啦！"**)  
  
 *#类方法* @classmethod  
 **def** class\_fun(cls):  
 print(**"A类的类方法被调用啦！"**)  
  
 *#静态方法* @staticmethod  
 **def** static\_fun():  
 print(**"A类的静态方法被调用啦！"**)  
  
  
a = A()  
  
print(**'======我是一条美丽的分割线======'**)  
#查看规律：  
# 1.先用类分别调用三种方法  
**try**:  
 A.fun()  
**except** TypeError:  
 print(**"类无法调用实例方法！"**)  
A.class\_fun()  
A.static\_fun()  
# 2.再用由类实例化过来的对象调用三种方法  
print(**'======我是一条美丽的分割线======'**)  
a.fun()  
a.class\_fun()  
a.static\_fun()  
  
由此 我们可以得出以下规律：  
 实例方法 类方法 静态方法  
a = A() a.fun() a.class\_fun() a.static\_fun()  
A 不可用 A.class\_fun() A.static\_fun()  
即：类方法，静态方法，类和由类实例化出来的对象，皆可直接调用！  
对于实例方法，只能由类实例化的对象进行调用，类本身无法直接调用实例方法

1. **Python中如何调用基类的成员方法？有几种方式？有什么区别吗？**

Python中调用父类的两种方式  
 示例代码——单继承：  
**class** A(object):  
 **def** \_\_init\_\_(self):  
 self.name = **'A:name'** print(**'A:\_\_init\_\_'**)  
 **def** fun(self):  
 print(**"A:fun"**)  
**class** B(A):  
 **def** \_\_init\_\_(self):  
 print(**"B:\_\_init\_\_"**)  
 A.\_\_init\_\_(self) *# 使用类名直接调用* super(B, self).\_\_init\_\_() *# 使用super关键字调用* **def** fun(self):  
 print(**"B:fun"**)  
 A.fun(self)  
 super(B, self).fun()  
 print(self.name)  
b = B()  
 总结：两种调用父类方法的方式其实都可行，区别在于，直接使用父类名称进行调用，父类方法只会被调用一次，使用super函数调用就是当前子类的基类的方法，可能调用不止一次。

1. **Python中如何调用基类的构造方法？**

答案同上。

1. **简单说一说Python中的经典类和新式类**

经典类：一种没有继承的类，所有的类型都是type类型，如果经典类作为父类，子类调用父类构造函数会报错。  
 新式类：每一个类都继承自一个基类，默认继承自object，子类可以调用基类的构造函数，所有类都有一个公共的祖先类object。

1. **什么叫二义性？**

二义性：Python因为支持多继承，而多继承的编程语言往往存在二义性问题。二义性问题分为两种：  
 1.有两个基类A和B，A和B中都定义了方法f(),C类继承了A类和B类，那么调用C类的f()时出现不确定情况。(魔鬼三角继承)  
 2. 有一个基类A，定义了方法f()，B类和C类继承了A类（的f()方法），D类继承了B和C类，那么出现一个问题，D不知道应该继承B的f()方法还是C的f()方法。(恐怖菱形继承)

1. **除了Python之外还有什么编程语言是支持多继承的？你知道Java中如何解决多继承的问题吗？**

支持多继承的编程语言：C++

Java本质上不支持多继承，但可以依托接口实现多继承。

1. **详细解释：方法解析顺序MRO**

参见链接：[Python3新式类的MRO算法——C3算法.py](https://github.com/La0bALanG/files/blob/master/%E4%B8%B2%E8%AE%B2%E8%B5%84%E6%96%99/%E8%80%81%E5%AE%89%E4%B8%B2%E8%AE%B2%E8%B5%84%E6%96%996%E2%80%94%E2%80%94Python3%E6%96%B0%E5%BC%8F%E7%B1%BB%E7%9A%84MRO%E7%AE%97%E6%B3%95%E2%80%94%E2%80%94C3%E7%AE%97%E6%B3)

地址：

https://github.com/La0bALanG/files/blob/master/%E4%B8%B2%E8%AE%B2%E8%B5%84%E6%96%99/%E8%80%81%E5%AE%89%E4%B8%B2%E8%AE%B2%E8%B5%84%E6%96%996%E2%80%94%E2%80%94Python3%E6%96%B0%E5%BC%8F%E7%B1%BB%E7%9A%84MRO%E7%AE%97%E6%B3%95%E2%80%94%E2%80%94C3%E7%AE%97%E6%B3%95.py.py

1. **Python3中如何表示MRO呢？**

Python3中，将MRO封装为任意一个自定义类的属性，所以，通过：

类名.\_\_mro\_\_即可查看任意一级最终子类的方法解析顺序。

1. **\_\_new\_\_ 和\_\_init\_\_方法的区别在哪呢？**

\_\_init\_\_方法是当对象创建后,实例化对象自动执行的一种方法,而\_\_new\_\_方法和\_\_init\_\_方法不同,可以通俗点理解\_\_init\_\_方法在\_\_new\_\_方法执行后再执行.

\_\_new\_\_方法是用来创建对象的,\_\_new\_\_方法需要有一个返回值,这个返回值表示创建出来的对象的引用.

\_\_init\_\_方法是刚刚创建出来的对象的应用.

1. **什么叫抽象类？什么叫抽象方法？Python中如何创建一个抽象类？如何创建抽象方法？实际开发中，抽象类的作用是什么？**  
    1.抽象方法：不包含任何可实现代码的方法就叫做抽象方法，即一个方法中没有任何一个方法体  
    2.定义：抽象类是包含抽象方法的类，只能在其子类中实现抽象方法的代码  
    3.抽象方法的定义  
    1.导入abc模块  
    import abc  
    2.如果需要单个导入的话，一般我们只需要导入ABCMeta和abstractmethod就 行  
    from abc import ABCMeta,abstractmethod  
    3.将元类设置为ABCMeta  
    \_\_metaclass\_\_ = ABCMeta  
    4.抽象方法前加装饰器abstractmethod  
    @abstractmethod  
    4.抽象类的特点：  
    1.要定义但是并不完整的实现所有方法  
    2.基本的大概意思其实就是父类  
    3.父类需要明确表示出哪些方法的特征，  
    5.需要使用抽象类的地方  
    1.用作父类  
    2.用作检验实例类型  
    3.用作抛出异常说明
2. **详细解释为什么要进行运算符重载，函数重写？使用Python编程实现逻辑运算符重载与getattr和setattr函数重写**

函数重载主要是为了解决两个问题。1。可变参数类型。2。可变参数个数。另外，一个基本的设计原则是，仅仅当两个函数除了参数类型和参数个数不同以外，其功能是完全相同的，此时才使用函数重载，如果两个函数的功能其实不同，那么不应当使用重载，而应当使用一个名字不同的函数

具体代码举例参见链接：[overwrite.py](https://github.com/La0bALanG/files/blob/master/%E4%B8%B2%E8%AE%B2%E8%B5%84%E6%96%99/%E8%80%81%E5%AE%89%E4%B8%B2%E8%AE%B2%E8%B5%84%E6%96%995%E2%80%94%E2%80%94overwrite.py)

地址：

https://github.com/La0bALanG/files/blob/master/%E4%B8%B2%E8%AE%B2%E8%B5%84%E6%96%99/%E8%80%81%E5%AE%89%E4%B8%B2%E8%AE%B2%E8%B5%84%E6%96%995%E2%80%94%E2%80%94overwrite.py

1. **Python中都有哪些‘魔术方法’？**

双下划线开头结尾的其实就是魔术方法

1. **请问类是一个对象吗？手动使用type()来创建一个类对象，需要传递几个参数？每个参数的含义是什么？使用type()函数创建类对象时如何添加成员方法？**

类，也是一个对象，因为类是由元类创建的，类拥有自己的属性和方法。手动使用type()函数来创建一个类对象，需要传递三个参数

此时type函数语法：

类名 = type(String,tuple,dict)

参数解释：  
 String：字符型参数，类的名称  
 tuple:元组，继承关系  
 dict：字典，定义属性及方法

如果需要添加成员方法，可先编写好一个函数，然后给第三个参数字典内添加一个键值对，键名为方法名称，键值为该函数。