面向对象编程（python）【oop】

什么是对象：万物皆对象。

对象是具体的物体：拥有属性，行为，把很多零散的东西封装成一个整体。

python是一门十分彻底的面向对象编程的语言，python中所有的int，float，True..list等都属于对象类属性

基本理论

面向对象和面向过程的区别

面向过程：在解决问题时关注解决问题的每一个步骤

·面向对象：关注的时解决问题所需要的对象，谁可以解决这个问题就去找谁。

·面向对象是面向过程的一种封装

·面向对象最关键的步骤是划分对象，要把哪些功能划分到哪些对象里面，同时确定对象的属性和行为

·根据抽象的类，生产具体的对象。 类包括：名称+静态属性（属性）+动态行为（方法）

·类是抽象的，对象是具体的。比如：汽车类，具体后就是奥迪，宝马，奔驰对象。

·类到对象是一个实例化的过程，对象可以抽象出类，但是本质上类也是一个属性

在python中的实践

- class 类名： （pass，如果没东西就pass表示空）

```
class Money:  #不需要加小括号
    pass
#类名的标准必须遵循大头标识规定，一开头的标识必须是大写
#类名后面加小括号表示继承

#根据这个类，实例化一个对象
one = Money()#这个one就是一个对象
one.__class__ = Money
```

- python中的类又分为经典类/新式类
- 在线绘制流程图的软件：ProcessOn（百度搜索在线绘图）
  -

```
__name__ = Money # 类名叫Money，同时Money也是一个变量
```

- 属性是属于某个对象的特性，一定要指定某个变量，同时属性只能通过变量来访问
  - 例如：zhangsan.age = 000，才可以进行访问。
  - 同时属性必须拥有宿主，这是与变量的区别

- 属性又分为两种：

  1.对象属性：

  2.类属性：

- 如何给对象增加属性？
  - 1.直接通过对象动态添加

```
class Person:
    pass

p = Person()
p.age = 18
print(p.age)
```

  - 对象._ *dict* _，表示显示出目前对象中的所有属性值，输出一个字典类型的键值对
  - 如果修改的化，是重新拿一快地址来进行储存，append是追加，则不会增加地址。
- 如何删除一个对象的属性？
  - 和删除变量是一样的操作，del
  - 不同的对象不能访问对方的属性

类名也可以作为一个对象来定义属性，如下：

```
class Person:
    pass

p = Person()
p.age = 18
# print(p.age)
# print(p.__dict__)

Person.Lbw = 666
print(Person.Lbw)
print(Person.__dict__)
```

定义也可以在class类后面的退格内部直接写。

对象会先从自身找属性，如果自身没有再去类对象中找，但是如果自己内部有，则优先从自己身上找。

修改一个类属性的值是重新开辟一段内存，而不是把原本内存指的值进行修改

删除属性，只能删除自己的直系属性，不可以向上，能够向上级传递的只有查找！

- 类里面的 _*dict* _ 是不可以更改的，是只读的一个字典。(可以通过setattr的方法进行修改)
- 对象里面的——dict——是可以进行更改的。
- 类属性被各个对象所共享。

- 

```
class Person:
    pass

##类别属性
Person.age =19
del Person.age

##对象属性    其二者的区别要搞清楚
p = Person()
p.age = 11
del p.age
```

- **如何限定对象可以添加的属性，防止太杂太乱**

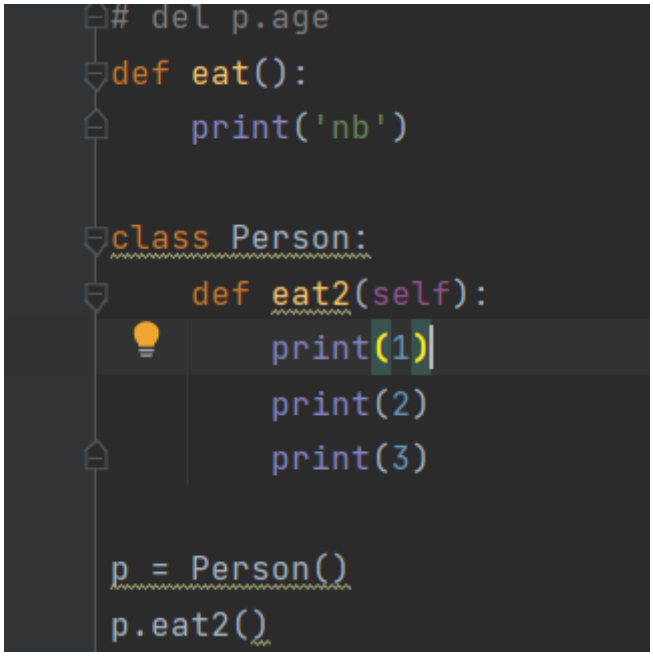
所以限制方法：

```
class Person:
    __slots__ = []#这个列表里面限定了你后面对象可以添加的属性，如果属性不再这个列表内部则会报错。
```

python面向对象中的方法

- 方法的概念：是描述一个目标的行为动作。比如描述一个人怎么吃、怎么喝、怎么玩
- 目标可以是一个类，也可以是一个对象。方法都是用来描述目标的。

定义函数和定义方法的区别：



```
# del p.age
def eat():
    print('nb')


class Person:
    def eat2(self):
        print(1)
        print(2)
        print(3)


p = Person()
p.eat2()
```

定义方法时会自动补全内部的self。

同时调用方法的时候，需要先定义一个对象，通过对象来使用方法

- 方法的划分：
  1. == 实例方法：类--->通过实例化---->具体对象，反映出类到对象的过程，所以类也可以被称之为对象，对象被称之为实例！！！！ ==　　所以从现在开始，类被称之为对象，而原本的对象被称之为实例。

     默认第一个参数需要接受到一个实例
  2. 类方法：默认第一个参数需要接受到一个类
  3. 静态方法：什么也不默认接受
  4. 注意：

```
class Person:
    def eat(self):  #定义实例方法
        print(1)
        #所以实例方法的调用要先定义实例
        p = Person()
        p.eat()

    @classmethod  #定义类方法 ，这行艾特也要写
    def leifangfa(cls):

    @staticmethod
    def jingtaifangfa（）：#静态方法的定义，括号内不用补全
```

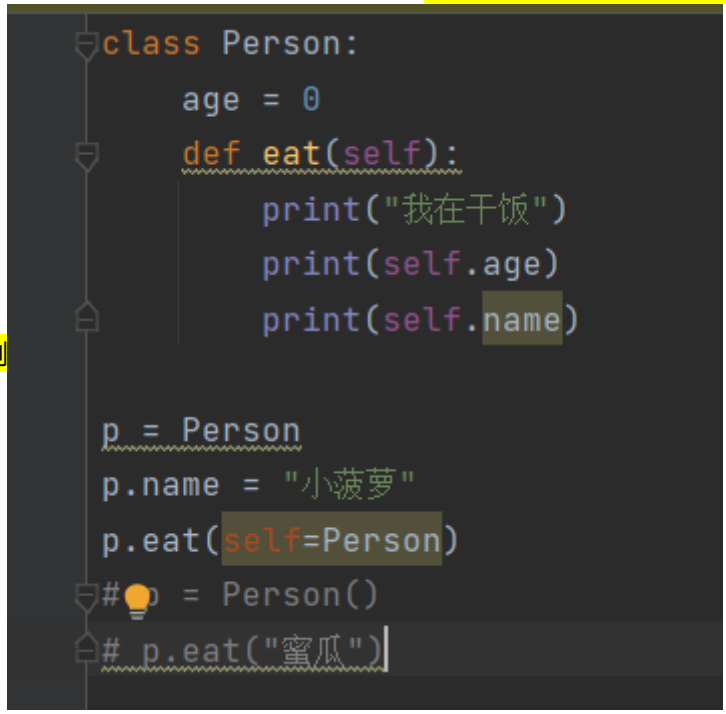
- 三种方法的定义都存储在的是类对象里面的dict里面，不存在在实例对象里面！！
- 实例方法的调用方式：
  - 标准调用：先创建实例，然后编译器会自动帮我们传第一个参数self，我们不需要传第一个参数，而如果我们需要传其他参数，直接在self后面加就可以

  - 其他调用
    - 直接当成函数调用，比如直接`Person.eat()`但是这种方法则需要传入两个参数，要把self也放进去
    - 或者func = Person.eat（），和上面一样的，只是做了一个赋值操作。
- 类方法的语法与调用方式：
  - 需要有一个装饰器`@classmethod`，之后在定义方法时则会自动填充cls这个参数
  - 类方法可以直接通过类调用，也可以通过实例进行调用，但是其对应的实例会被忽略，类会被保留
- 衍生类（子类继承父类）

  `class A(Person):`子类A继承了Person的类

使用衍生类调用类方法时，内部的cls参数会编程衍生类：A的参数，会进行改变

- 静态方法，调用方法和上面一样，没有第一个输入参数的要求
- 补充：类对应实例是一对多的关系，实例对应类是多对一的关系，可以通过一个实例来找到类，但是不能通过一个类来找到实例



可以通过self实例属性来访问类里面的内容

但是如果你用类方法的话就找不到下面实例里面的name小菠萝！！！

如何查找一个对象对应的类？`__class__`

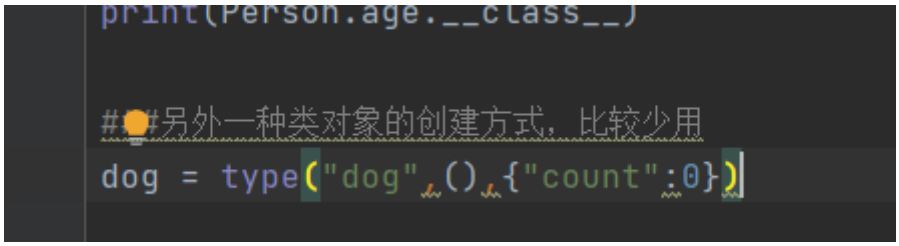
补充

类相关的补充

1. 元类：创建类对象的类，类对象是由另外一个类创建出来的->而这就称之为元类。

如何查找元类：`num.__class__.__class__`,而其实这些类（int，str...）都是由一个非常NB的类创建出来的，他就是type（元类）；

2.用type创建类（type还可以检测类型）



这种方法在继承和多态中应用比较多！！

元类的应用后面会具体补充，不急

类的描述文本

通过三个双引号对`"""66666"""`添加对类的描述

通过`help(类名)`来查看我们对这个类的描述

如何生成项目文档：P38-P40

使用内置模块pydoc，抽离出文件内的所有注释，还有一些外置的工具模块，比如Sphink、epydoc等，需要使用的时候再看。

终端中python -m 表示以脚本形式运行python

- 私有化属性的目的：减少访问的方式，增加安全性
- 私有化属性的方法：

三种类型的属性(Attribute)再四个位置的调用方式

前两种见ipad笔记

`__x`表示私有属性：类的内部可以访问、子类内部不能访问、模块内部其他位置也不行、其他模块中用`import 类名 print("类名+__a")`可以访问，但是用`from leiming import *`就不可以

私有属性的实现机制：

略，而且私有尽量不要去访问，不太好哦。

只有在类的内部创建才可以创建一个私有化属性，在实例内部无法创建，而且创建出来的私有化属性是带类名的，比如：`'_Person__age'`

私有化属性的应用场景

创建方法

```
class Person:
    def __init__(self):  #表示系统内置的方法，称之为初始化方法
        self.age = 18
        ###主要作用是：当我们创建一个实例对象之后会自动调用这个方法，来初始化这个对象
p1 = Person()
p1 = 19

p2 = Person()


p3 = Person()


以上三个实例在创建之后都会先运行`__init__(self):`内部的东西进行初始化
```

判断一个数据是不是一个类型：

if isinstance（value，int），判断输入的value是不是int类型。很好用，限制输入。

在变量后面加_的目的就是与系统内部设定的字符做一个区分，比如class 和 `class_`

如果出现 `__xxx__` 这种一般表面是系统内置的功能。

- 只读属性：系统内部可以得到，但是外部只能得到他的状态，不能更改其值
  - 通过在初始化函数内设置为私有属性，禁止其读写操作。然后再打开其读属性，再内部创建一个方法然后返回这个私有属性。

```python
class Person:
    def __init__(self):
        self.__age = 18
    def readme(self):
        return self.__age
p1 = Person()
print(p1.readme())
```

在初始化和方法内部都要加上前缀self，否则会报错。

- 优化后的效果
-

```python
class Person:
    def __init__(self):
        self.__age = 18
    @property
    ## 这个装饰器的作用就是可以使用属性的方式，来使用这个方法。
    def readme(self):
        return self.__age
p1 = Person()
print(p1.readme)
```

在方法之前加上@property装饰器，可以直接实现只读功能，所以一般用第二个方法。

- property的作用：将一个属性的与另外一个属性及器对应的操作方法关联起来
- **经典类**和**新式类**两种类别
  - 经典类：没有继承（object）
  - 新式类：继承（object）
  - 父类（基类），子类。通过 `类名.__bases__` 的方式查看一个类别的基类是什么
    - 一般python3开始的类别默认是新式类。但是二点几版本的新式类必须在定义类的时候+（object）才可以定义一个新式类。
  - 概念补充：property也可以作为一个函数输入，并将两个方法关联起来赋予同一个方法
  -

```python
class Person:
    def __init__(self):
        self.__age = 18

    def get_age(self):
        return self.__age

    def set_age(self,value):
        self.__age = value

    age = property(get_age,set_age)

p1 = Person()
p1.age = 80
print(p1.age)
```

装饰器还有设置的，还有写入的，具体见装饰器的文本。

- 经典类了解了解就行。
  - 如果不支持中文，需要在文件最上面写上`_*_ encoding:utf-8 _*_`
  - shift+table 整体代码向前缩进。
- 方案2：

  当我们通过：实例.属性 = 值时，给一个实例增加一个属性，或者时修改属性值的时候都会调用下面这个方法：

```python
class Person:
    def __setattr__(self,key,value):
        print(key,value)

p1 = Person()
p1.age = 18

###则会打印出
```

常用的内置属性

- 类属性
  - `__dict__`:查看类里面的所有属性+方法+对象
  - `__bases__`:类的所有父类构成的元组 ，说明python支持多继承
  - `__doc__`:类的文档字符串/就是那个三对分号里面的注释。
  - `__name__`:类名
  - `__dict__`:类定义所在的模块
- 实例属性
  - `__class__`:实例对应的类

方法相关补充

1. 私有化方法的定义

```
class Person:
    __age = 18

    def __run(self):
        print("pao")
```

和私有化属性一样在前面加两个下划线就可以。

一般在系统内部的dict里面对于的方法是带上类的名称的：`_Person__age`,所以可以直接通过全程来覆盖上一个方法也是可以的。

2. 方法内置特殊方法
    - 生命周期方法
    - 其他内置方法
        - 

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

p1 = Person(22,"xiaoqao")
print(p1.age)
print(p1.name)

p2 = Person(22,"xiaoni")
print(p2.age)
print(p2.name)
```

一种可以给每个实例给不同初始值的解决方法。

        - 信息格式化操作：
            - `__str__`方法：`def __str__(self):`通过这个方法对print（实例），输出内容进行格式化
            - `__repr__`方法：和上面的方法差不多，主要面向开发人员，所以用的比较少。获取实例的本质信息，比如地址，类型等等
        - 调用操作：`__call__`方法：作用是使得对象具备当作函数来调用的能力。

- 偏函数的使用方法：

```
def createPen(p_color,p_type):
    print("create a %s type pen , it is %s" % (p_type,p_color))


import functools

gangbiFunc = functools.partial(createPen, p_type="港币")

gangbiFunc("red")
```

用面向对象的思路：

```
##面向对象的使用方法：
class Penfactory:
    def __init__(self,p_type):
        self.p_type = p_type

    def __call__(self, p_color):
        print("create a %s type pen , it is %s" % (self.p_type,p_color))

gangbiF = Penfactory("钢笔")
gangbiF("红色")

qianbiF = Penfactory("铅笔")
qianbiF("绿色")
```

```
C:\ProgramData\Anaconda3\python.exe C:/Users/高涵/PycharmProjects/inference/inference.py
create a 钢笔 type pen , it is 红色
create a 铅笔 type pen , it is 绿色
```

3.索引操作：对一个实例对象进行索引操作，然他具有列表或者是字典的属性要求。

设置一个项目：`def __setitem__(self, key, value):`

增：`def __setitem__(self, key, value):`

删：`def __delitem__(self, key, value):`

改：`def __getitem__(self, key, value):`

查：`def __setitem__(self, key, value):`

有以上的方法之后，可以直接用一个实例进行键值对的添加，必须要有上面的几个实例方法才可以。

4.切片操作：

和索引操作里面的方法相同。可以和初始化init方法绑定一个初始值，然后进行赋值操作，如果忘记了回去看62-64视频

5.类定义实例的大小比较：

`def __eq__(self,other):`就可以进行比较是否相等，比较规则自己设定，设定在这个方法内部。

`def __gt__(self,other)`:判断是否大于操作

`def __lt__(self,other)`:判断是否小于操作

le小于等于

如何通过装饰器自动地推到出其他的功能?

```
#首先引入包
import functools

@functools.total_ordering
#则会自动补全别的功能，6种比较方法
```

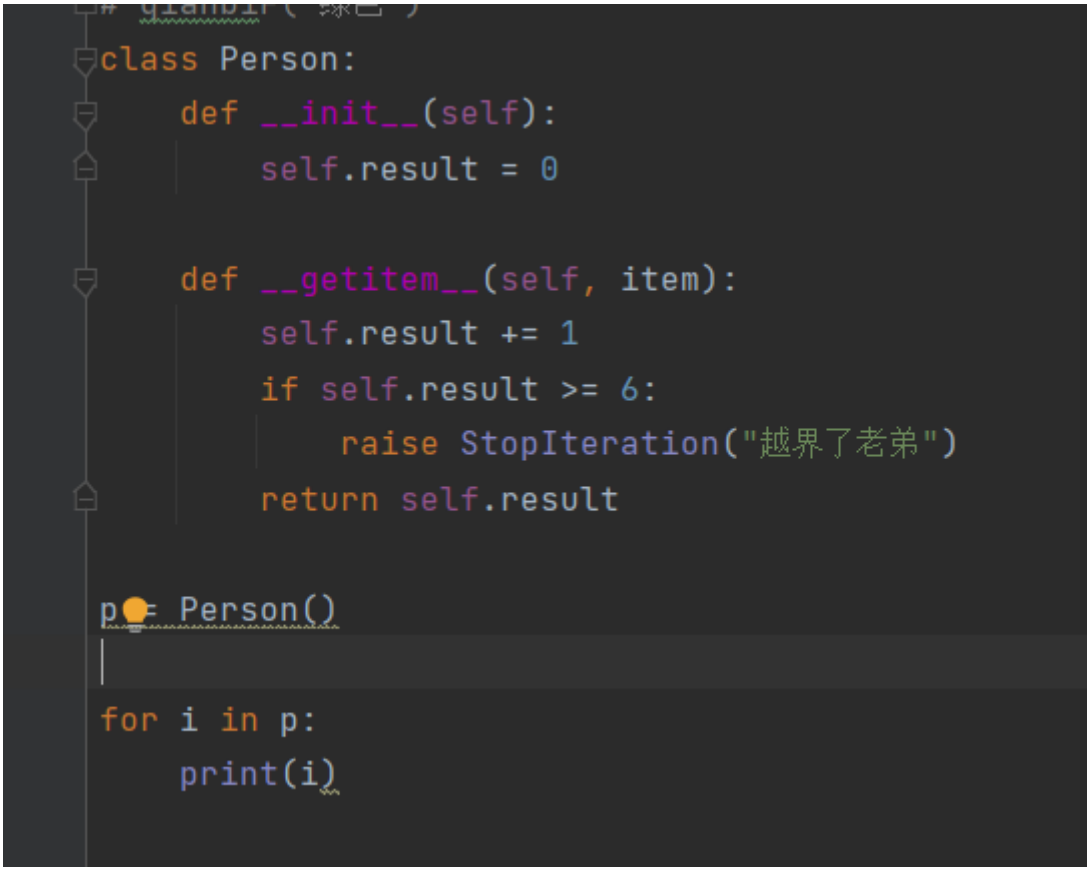
6.上下文环境中的布尔值:

if 8:

非零即真，所以都会进行打印。但是你要是定义了`def __bool__(self):+return放回值`，下面的实例调用就会根据你的返回值进行新的调用。

7.如何对对象可以进行for in遍历循环访问?

- 方式1



raise 表示抛出一个异常情况。

- 方式2:

```
def __iter__(self):
    ##这个方法的优先级大于getitem这个方法，所以会先调用iter
    #这个方法是返回一个迭代器，通过next函数进行迭代。
        #  def __next__(self):  所以还要定义这么一个方法进行迭代
```

遍历操作P68-73，完全没看懂，有空回去看看

面向对象的描述器

- 描述器：是一个描述属性操作的对象，多个方法封装到一个对象内，就称之为描述器。
- 可以对传递过来的数据进行数据过滤和数据验证。
- 描述器可以把你之前定义好的类里面的方法拿出来，放在一个描述器里面，后面就可以对找个描述器进行增删改查
-

```
class Person：里面
def del_age:
    ....
def get_age:
    ...
def set_age:
    ...
age = property(get_age, set_age, del_age)


p = Person()
p.age = 19#就可以直接进行增加
```

- 也可以使用装饰器和描述器一样但是用：@property

装饰器--类实现

```
class check:
    def __init__(self, func):
        self.f = func #首先要先记得保存这个方法
    def __call__(self,*args,**kwargs):
        self.f() #定义一个实例之后记得调用

@check
def fashuoshuo():
    print("fashuoshuo")
fashuoshuo = check(fashuoshuo)
```

python的生命周期，以及周期方法

- 生命周期是值一个对象从诞生到消亡的过程，当一个对象被创建的时候，会在内存中分配相应的内存空间进行存储，但对象不再使用时，为了节约内存就会把这个对象释放
- 通过`def __new__`来拦截一个实例的创建
- `def __del__`表示一个对象被释放之后马上会调用这个方法，可以用来监听生命周期
- python 为了提升整体的性能，将整数以及一些短小的字符进行缓存，不管调用多少次，占用的都是同一个地址。
- 查看一个对象（函数）被引用的次数

```
import sys
sys.getrefcount(对象)#会比正常次数大一
```

综合案例：

一个方法可以用多个装饰器，而装饰器的顺序按照你自己程序需要运行的先后顺利来设定。

链式编程思路，在每一个方法的最后return self

之后在调用实例时比如:

```
c1 = caculator()
c1.jia(4).jian(5).cheng(3).chu(7).show()
```

非常的方便啊

资源的继承

继承仅仅是资源的使用权，并没有把父类的属性或者方法拷贝过来或者拿过来

父类的私有属性__不可以被继承

但是内置属性可以被继承

更改子类里面的属性，并不会影响父类，但是父类更改会影响子类

多继承遵循单继承链原则

在python3.x之后的资源标准原则的方案：

只有新式类，没有经典类，采用的都是C3算法，采用MRO原则

C3算法：L[object] = [object] 查看一个类，就是一个类对于的列表

L（子类（父类1，父类2）） = [子类] + merge(L(父类1)，L（父类2），[父类1，父类2]) #合并列表

merge函数就是检测一个列表是不是有后面函数出现的列表

资源的覆盖

并不是覆盖，只是由于资源访问的优先级导致了访问出来到底是哪个类的类型

也可以重写方法

资源的累加：在被覆盖的方法基础之上新增内容，可以在继承的子类里面增加自己的方法

多态

一个类所延申的多种形态/调用时的多种形态，同一个类里面的功能面向不同的对象可能不同，比如小猫的叫和小狗的叫可能不同。不同的对象，调用相同的方法，表现出不同的功能。

但是python中没有真正意义上的多态，别的语言有。

- 抽象类：上面的那个animal
- 抽象方法：就是上面的那个叫，如果使用抽象方法必须全部有定义。

Python面向对象的设计原则

SOLID原则

Single Responsibility Pribciple原则（单一职责原则）

一个类只负责一个职责

O开放封闭式原则，对扩张开放对修改关闭

I接口原则

D依赖倒置原则