面向对象基础（上）

复习

数学运算

abs( ) round( )

pow( ) divmod( ) max( ) min( ) sum( )

eval( )

类型转换

int( ) float( ) str( )

ord( ) chr( ) bool( )

bin( ) hex( ) oct( )

list( ) tuple( )

dict( ) bytes( )

序列操作

all( ) any(sorted() reversed( )

range( ) zip( ) enumerate( )

三元运算

result = 值 1 if 条件 else 值 2

set 集合

无序且不重复的元素集合。

add( ) clear( )

difference( ) intersection( ) union( )

pop( ) discard( ) update( )

导入

Python 从设计之初就已经是一门面向对象的语言，正因为如此，在 Python 中创建一个类和对象是很容易的。本章节我们将详细介绍 Python 的面向对象编程。

目录

目标

1、通过明确面向对象语言的基本特征，在头脑里形成基本的面向对象的概念（重点、难点）

2、通过定义类和对象，对事物进行抽象化（重点）

3、使用__init__方法、self 属性快速初始化属性（重点、难点）

4、使用魔法方法轻松增加类的功能（重点）

一、面向对象介绍

1、概述

面向过程：根据业务逻辑从上到下写代码

函数式：将某功能代码封装到函数中，日后便无需重复编写，仅调用函数即可

面向对象编程：将数据与函数绑定到一起，进行封装，这样能够更快速的开发程序，减少了重复代码的重写过程

2、面向过程、函数式、面向对象的比较与区分

面向过程编程最易被初学者接受，其往往用一长段代码来实现指定功能，开发过程的思路是将数据与函数按照执行的逻辑顺序组织在一起，数据与函数分开考虑。

我们看看发送告警面向过程实现，发现有大量的重复代码。

```
while True:
    if cpu 利用率 > 90%:
        # 发送邮件提醒
        连接邮件服务器
        发送邮件
        关闭连接
    if 硬盘利用率 > 90%:
        # 发送邮件提醒
        连接邮件服务器
        发送邮件
        关闭连接
    if 内存占用 > 90%:
        # 发送邮件提醒
        连接邮件服务器
        发送邮件
        关闭连接
```

后来我们学习了函数之后，升级下代码，采用函数式编程。

```
def 发送邮件(内容)
    # 发送邮件提醒
    连接邮箱服务器
    发送邮件
    关闭连接

while True:

if cpu 利用率 > 90 %:
    发送邮件('CPU 报警')

if 硬盘使用空间 > 90 %:
    发送邮件('硬盘报警')

if 内存占用 > 80 %:
    发送邮件('内存报警')
```

今天我们来学习一种新的编程方式：面向对象编程。

不同方式解决洗车问题；

(1)第一种洗车的方式

1、自己上某宝买洗车工具。

2、根据自己的不多的预算买，抹布，水泵，清洁剂。

3、跟店小二狂砍一个小时，便宜了 2 块钱。

4、等了一周，买的东西到了，开始洗车

5、洗到一半，水泵坏了，泡泡还没洗干净

(2)第二种洗车的方式

1、找一家靠谱的洗车店

2、给钱洗车

面向过程和面向对象都是一种解决实际问题的思路。

洗车第一种方式：强调的是步骤、过程、每一步都是自己亲自去实现的,这种解决问题的思路我们就叫做面向过程

洗车第二种的方式：强调的是洗车店，由洗车店去帮我们洗车，过程不用我们自己去操心，对我们而言,我们并不必亲自实现整个步骤只需要调用洗车店就可以解决问题，这种解决问题的思路就是面向对象。

面向对象：按人们认识客观世界的系统思维方式,采用基于对象(实体) 的概念建立模型,模拟客观世界分析、设 计、实现软件的办法。

面向对象编程(Object Oriented Programming-OOP)是一种解决软件复用的设计和编程方法。这种方法把软件系统中相近相似的操作逻辑和操作 应用数据、状态,以类的型式描述出来,以对象实例的形式在软件系统中复用,以达到提高软件开发效率的作用。

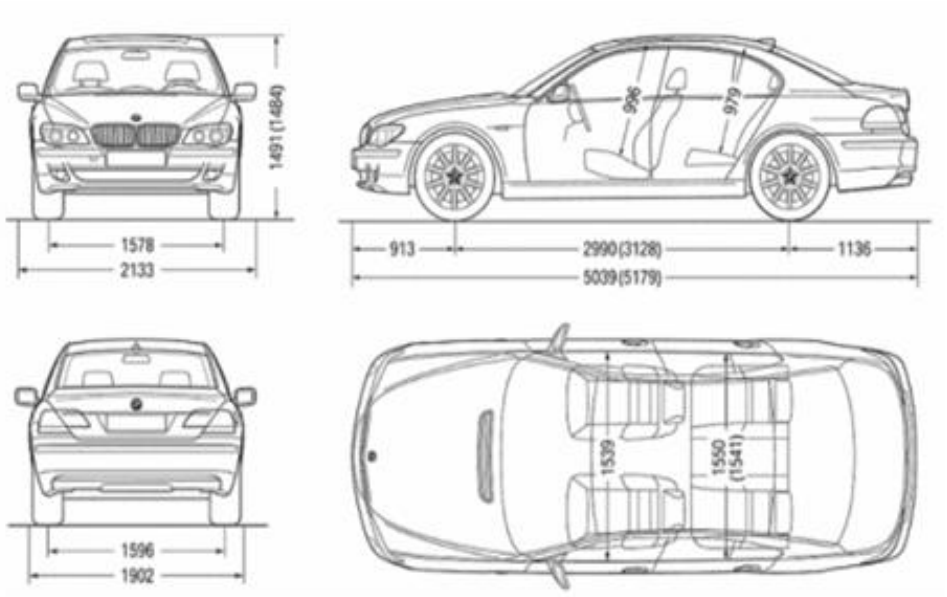
二、类和对象的概念

1、类和对象

- 类和对象是面向对象编程中重要的概念
- 类就是一个模板，模板里可以包含多个函数，函数里实现一些功能
- 对象则是根据模板创建的实例，通过实例对象可以执行类中的函数

类相当于制造汽车的图纸，用这个图纸制造的汽车相当于对象。

类



对象

2、类的组成部分

类(Class)由 3 个部分构成

- 类的名称: 类名
- 类的属性: 一组数据
- 类的方法: 允许对进行操作的方法 (行为)

创建一个人类：

- 事物名称(类名):人(Person)
- 属性:身高(height)、年龄(age)
- 方法：吃(eat),跑(run)...

3、类的抽象

拥有相同(或者类似)属性和行为的对象都可以抽像出一个类

（1）、小明开着他家的宝马去逛街

小明 ->可以抽象中人类

宝马 ->可以抽象出车类

（2）坦克大战抽象类：

抽象坦克大战中的类：

坦克：

- 类名：Tank
- 属性：血量 类型
- 方法：发射子弹

墙：

- 类名: Wall
- 属性：血量 类型
- 方法：阻挡

子弹：

- 类名：Bullet
- 属性：杀伤力

三、定义类和对象

1、定义类

定义类格式：

```
#创建一个类
class Foo(object):
    方法列表
```

示例：

（1）创建一个人类：

```
# 创建一个类
class Person(object):
    name = 'xiaohong'
    age = 18
    def eat(self):
        print('吃饭')
```

（2）创建一个车类：

```
# 创建一个车类
class Car(object):
    # 属性
    colour = 'red'
    # 方法
    def run(self):
        print('车在跑')
```

注意：类名采用大坨峰方式命名。

2、创建对象

格式：

```
对象名 = 类名()
```

示例：

创建一个 Car 对象，bmw 就是 Car 的对象，对象可以访问属性，调用方法。

```
# 创建一个车类
class Car(object):
    # 属性
    colour = 'red'
    # 方法
    def run(self):
        print('车在跑')

bmw = Car() # 创建对象
bmw.run() # 调用 run 方法
print(bmw.colour)  # 访问类属性
```

四、实例方法与属性

1、实例方法

在类的内部，使用 def 关键字可以定义一个实例方法，与一般函数定义不同，类方法必须包含参数 self,且为第一个参数

```
class Animal(object):

    # 实例方法，使用 def 关键字定义，第一个形参默认传实例对象本身，一般使用 self 作为第一个参数
    def test(self):
        print("我是实例方法")

        # 一个类里面可以有多个实例方法

    def show(self):
        print("Animal.show")
```

2、属性

属性：在类里面定义的变量。定义在类里面，方法外面的属性称之为类属性，定义在方法里面使用 self 引用的属性称之为实例属性

```
class Animal(object):

    colour = '白色'      # 类属性

    def __init__(self):
        self.name = '旺财'    # 实例属性
```

```
    # 实例方法，使用 def 关键字定义，第一个形参默认传实例对象本身，一般使用 self 作为第一个参数
    def test(self):
        print("我是实例方法")

    # 一个类里面可以有多个实例方法
    def show(self):
        print("Animal.show")
```

属性在后面章节中会详细讲解，这里主要先了解属性这个概念。

五、__init__方法

1、分析问题

创建一个 Animal 类，并实例化一个 dog 对象和 cat 对象。

```
class Animal(object):
    def eat(self):
        print('吃饭')


dog = Animal()  # 创建一个对象
dog.colour = 'red'  # 添加属性
dog.name = '旺财'  # 添加属性
print(dog.name)

cat = Animal()  # 创建一个对象
cat.colour = 'black'  # 添加属性
cat.name = '小花'  # 添加属性
print(cat.name)
```

每创建一个对象都要添加属性，试想如果再创建一个对象的话，肯定也需要进行添加属性，显然这样做很费事，那么有没有办法能够在创建对象的时候，就顺便对象的属性给设置呢？

2、__init__(self) 方法

__init__(self) 方法初始化方法，实例化对象的时候自动调用，完成一些初始化设置

```
# 创建一个动物类
class Animal(object):
    # 创建一个初始化方法
    def __init__(self):
        self.name = '旺财'
      self.colour = '黄色'
dog = Animal()  # 实例化对象
print(dog.name)  # 访问属性
print(dog.colour) # 访问 colour 属性
```

注意：_init_()方法在创建对象的时候自动调用。

3、 __init__传参

如果 init 方法里面的属性固定了，每个类创建出来的对象属性都一样，这个时候我们是不是考虑将属性当参数在实例化对象的时候传进去，让类更通用？

```
# 创建一个动物类
class Animal(object):
    # 创建一个初始化方法
    def __init__(self,name,colour):
        self.name = name
      self.colour = colour
dog = Animal('旺财', 'white')  # 实例化对象
print(dog.name)  # 访问属性
print(dog.colour) # 访问 colour 属性
```

```
dog = Animal('小花', 'black')  # 实例化对象
print(dog.name)  # 访问属性
print(dog.colour) # 访问 colour 属性
```

注意：

- _init_()方法，在创建一个对象时默认被调用，不需要手动调用
- _init_(self)中，默认有 1 个参数名字为 self，如果还需要传两个实参，那么应该写成_init_(self，x, y)。

六、理解 self

1、self 是什么

```
# 创建一个类
class Car(object):
    # 创建一个方法打印 self 的 id
    def getself(self):
        print('self=%s' % (id(self)))


bmw = Car()
print(id(bmw))
bmw.getself()
```

输出结果：

```
>>> # 创建一个类
... class Car(object):
...     # 创建一个方法打印 self 的 id
...     def getself(self):
...         print('self=%s'%(id(self)))
...
>>> bmw = Car()
>>> print(id(bmw))
140033867265696
>>> bmw.getself()
self=140033867265696
```

self 和对象指向同一个内存地址，可以认为 self 就是对象的引用。

2、self 传参问题

所谓的 self，可以理解为对象自己，某个对象调用其方法时，python 解释器会把这个对象作为第一个参数传递给 self，所以开发者只需要传递后面的参数即可

```
# 创建一个类
class Car(object):

    def __init__(self, name, colour):
        self.name = name
        self.colour = colour
```

```
        # 创建一个方法打印 self 的 id


def getself(self):
    print('self=%s' % (id(self)))


bmw = Car('宝马', '黑色')  # 实例化对象时，self 不需要开发者传参，python 自动将对象传递给 self
print(id(bmw))
bmw.getself()
```

七、魔法方法

1、概述

在 python 中，有一些内置好的特定的方法，方法名是“__xxx__”,在进行特定的操作时会自动被调用，这些方法称之为魔法方法。 下面介绍几种常见的魔法方法。

- __init__方法：构造初始化函数，在创建实例对象为其赋值时使用。
- __str__方法：在将对象转换成字符串 str(对象) 测试的时候，打印对象的信息。
- __new__方法：创建并返回一个实例对象，调用了一次，就会得到一个对象。
- __class__方法：获得已知对象的类 (对象.**class**)。
- __del__方法：对象在程序运行结束后进行垃圾回收的时候调用这个方法，来释放资源。
- ……

2、__str__方法

直接打印对象，输出结果只一串类似 id 地址的信息。

```
# 创建一个动物类
class Animal(object):
   # 创建一个初始化方法
    def __init__(self,name,colour):
        self.name = name
      self.colour = colour
```

```
dog = Animal('旺财', 'white')  # 实例化对象
print(dog)
```

输出结果：

```
>>> print(dog)
<__main__.Animal object at 0x7f5c2cea3ac8>
```

在类中定义给__str__方法

```
class Animal(object):

# 创建一个初始化方法
def __init__(self, name, colour):
    self.name = name
    self.colour = colour

def __str__(self):
    return '我的名字是%s,我的颜色为%s' % (self.name, self.colour)

dog = Animal('旺财', 'white')  # 实例化对象
print(dog)
```

输出结果：

```
我的名字是旺财,我的颜色为 white
```

定义了__str__方法，在打印对象的时候，会执行__str__方法。__str__方法只能 return 一个字符串。

八、案例-决战紫禁之巅

为了更好的理解面向对象编程，下面以“yy 决战紫禁之巅游戏”为案例.

1、问题分析

决战紫禁之巅有两个人物，西门吹雪以及叶孤城

属性：

- name 玩家的名字
- blood 玩家血量

方法：

- tong() 捅对方一刀,对方掉血 10 滴
- kanren() 砍对方一刀，对方掉血 15 滴
- chiyao() 吃一颗药，补血 10 滴
- __str__ 打印玩家状态。

2、实现步骤

（1）定义类，创建__init__方法

```
# 创建类
class Hero(object):
    #创建初始化方法
    def __init__(self, name):
        # 定义 name, blood 属性
        self.name = name
        self.blood = 100
```

（2）创建玩家技能方法：

```
def tong(self, enemy):
    """捅对方一刀"""
    enemy.blood -= 15
    info = '%s 捅了 %s 一刀' % (self.name, enemy.name)
    print(info)


def kanren(self, enemy):
    """砍对方一刀"""
    enemy.blood -= 15
    info = '%s 砍了 %s 一刀' % (self.name, enemy.name)
    print(info)


def chiyao(self):
    """吃药不血"""
    self.blood += 10
```

```
    info = '%s 吃了一颗补血药，加 10 滴血' % self.name
    print(info)
```

（3）创建__str__方法，输出玩家状态

```
def __str__(self):
    return '%s 还剩下 %s 血' %(self.name,self.blood)
```

（4）创建西门吹雪以及叶孤城两个人物

```
xm = Hero('西门吹雪')
ygc = Hero('叶孤城')
```

（5）两个开始互砍

```
xm = Hero('西门吹雪')
ygc = Hero('叶孤城')
xm.tong(ygc)  # 西门吹雪捅叶孤城一刀
print(ygc)  # 打印叶孤城的状态
print(xm)  # 打印西门吹雪的状态
print('*' * 20)

ygc.kanren(xm)  # 叶孤城砍西门一刀

print(ygc)  # 打印叶孤城的状态
print(xm)  # 打印西门吹雪的状态
print('*' * 20)

xm.chiyao()
print(ygc)  # 打印叶孤城的状态
print(xm)  # 打印西门吹雪的状态
```

3、运行结果

输出结果

```
C:\Users\Administrator\AppData\Local\Programs\Python\Python36-32\python.exe C:
西门吹雪 捅了 叶孤城 一刀
叶孤城 还剩下 85 血
西门吹雪 还剩下 100 血
**********
叶孤城 砍了 西门吹雪 一刀
叶孤城 还剩下 85 血
西门吹雪 还剩下 85 血
**********
西门吹雪吃了一颗补血药，加10滴血
叶孤城 还剩下 85 血
西门吹雪 还剩下 95 血
```

小结

类和对象的概念

类就是一个模板，模板里可以包含多个函数，函数里实现一些功能；

对象则是根据模板创建的实例，通过实例对象可以执行类中的函数。

定义类和对象

使用 class 语句来创建一个新类，class 之后为类的名称并以冒号结尾；

实例化类其他编程语言中一般用关键字 new，但是在 Python 中并没有这个关键字，类的实例化类似函数调用方式。

实例方法与属性

在类的内部，使用 def 关键字可以定义一个实例方法；

定义在类里面，方法外面的属性称之为类属性，定义在方法里面使用 self 引用的属性称之为实例属性。

__init__方法

构造初始化函数，在创建实例对象为其赋值时使用。

理解 self

self 和对象指向同一个内存地址，self 就是对象的引用。

魔法方法

在 python 中，有一些内置好的特定的方法；

方法名是“__xxx__”；

在进行特定的操作时会自动被调用。

课后作业

课后问答题

1、什么是类，什么是对象

2、python 中定义一个类的语法格式是什么

3、类(class)由哪三个部分构成

4、init 方法有什么作用，如何定义

5、方法中的"self"代表什么。

6、在类中定义 init 方法的时候第一个形参必须是 self 吗？self 可以用其他东西代替吗？

7、Python 面向对象中的魔法方法，是如何定义的，魔法方法需要开发人员去调用吗？

8、str 方法可以没有返回值，这句话是否正确？

9、查看下面代码，请写出有那些是属性，那些是实例方法。

```
class Person(object):
    foot = 2
    eye = 2
    mouth = 1
    def __init__(self,name,age):
        self.name = name
        self.age = age
        print("self=%s" % id(self))
    def run(self):
        print('飞快跑')
    def eat(self):
        print('吃饭')
xiaoming = Person('小明', 18)
print("xiaoming=%s" % id(xiaoming))
```

课后实操题

1、python 中如何通过类创建对象，请用代码举例说明。

2、如何在类中定义一个方法，请用代码举例说明。

3、定义一个水果类，然后通过水果类，创建苹果对象、橘子对象、西瓜对象并分别添加上颜色属性

4、请编写代码，验证 self 就是实例本身。

5、定义一个 Animal 类

(1)、使用 **init** 初始化方法为对象添加初始属性。如颜色，名字，年龄。

(2)、定义动物方法，如 run，eat 等方法。如调用 eat 方法时打印 xx 在吃东西就可以。

(3)、定义一个 **str** 方法，输出对象的所有属性。

6、请将课件上 决战紫荆之巅 重写一遍，并理解每一个方法的使用。